6-1974

The symbol-number representation method applied to compiling Chinese symbols.

Bill Wei-sung Chang
University of Tennessee

To the Graduate Council:

I am submitting herewith a thesis written by Bill Wei-sung Chang entitled "The symbol-number representation method applied to compiling Chinese symbols.." I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Science.

Robert M. Aiken, Major Professor

We have read this thesis and recommend its acceptance:

Accepted for the Council:
Carolyn R. Hodges

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

To the Graduate Council:

I am submitting herewith a thesis written by Bill Wei-sung Chang entitled "The Symbol-Number Representation Method Applied to Compiling Chinese Symbols." I recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Science.

Robert M. Aiken

Robert M. Aiken, Major Professor

We have read this thesis and recommend its acceptance:

Robert E Bollenheimer

Robert J Plemmons

Accepted for the Council:

Hilton A. Smith

Vice Chancellor
Graduate Studies and Research

THE SYMBOL-NUMBER REPRESENTATION METHOD APPLIED TO COMPILING CHINESE SYMBOLS

A Thesis

Presented for the

Master of Science

Degree

The University of Tennessee

Bill Wei-sung Chang

June 1974

ACKNOWLEDGMENTS

The author wishes to extend his sincere appreciation to his advisor, Dr. Robert M. Aiken, for his invaluable guidance, assistance, and encouragement during the course of this thesis. Also, the author expresses his gratitude to both Dr. Robert J. Plemmons and Dr. Robert E. Bodenheimer for their services on the Master's degree committee.

ABSTRACT

This thesis proposes a method to input the information in Chinese into the computer, and output in Chinese. The coding of Chinese word is designed practically.

The scanner, recognizer, and translator are illustrated with examples. From the appendices we can see this method can be carried out without difficulties.

TABLE OF CONTENTS

LIST OF TABLES

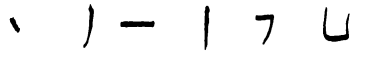
LIST OF FIGURES

CHAPTER I

INTRODUCTION

There is an evident and urgent need to use computers to solve many industrial and commercial problems. The Republic of China has bought computers from the United States since 1961. However these computers are not ideal, because one must input information in English, and obtain output in English. For example, in various types of accounting and bookkeeping systems, the original name and address of the depositer has to be translated into English before it can be read into the computer. In addition most of the Chinese people can not read English. For this reason, in 1970, the government assigned the Academic Sinnica and National Chiao Tung University to develop a computer system that people can communicate with directly in Chinese, which means that one can input Chinese characters and obtain the result in Chinese. A number of researchers have concentrated their efforts to build such a computer system; at present they haven't succeeded.

In other languages such as German, Spanish, and even Japanese this problem can be solved fairly easily, because their words are composed of letters. Letters in these languages can be coded and developed into their own special purpose computer language. But Chinese words are not composed of letters. Every word is formed with some elementary strokes, such as:

 . . . etc.

The arrangement of these elementary strokes is complicated. There are no regular rules to follow as to how one arranges these strokes. In addition there is no available method to code the words, therefore there is no simple method to input the word into the computer. Also it is not possible to print the output in Chinese. Thus the core of this problem involves the input and output. The solution to the problem of input and output is embedded in how to code the words.

Several ideas have been suggested as to how to solve the problem of using Chinese characters as input and output. One idea is to use the plotter to write the output in Chinese. This direct graphical output device allows the user to put a point at any location on the output page, and to move a pen any direction for a given distance. The speed is about 1 to 10 inches per second. The time needed to generate a word is of the order of second, and a subroutine can only generate one word. If we want to output a document of 1000 words we have to call the subroutine 1000 times. For a high speed computer the output will waste too much time. Also this idea can only solve the problem of output. It does not solve the entire problem. (reference 2, p-121)

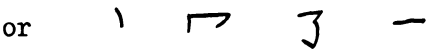
A second idea is to parse every word into elementary strokes or groups of elementary strokes. The usual way by which researchers code these strokes and groups of strokes is to represent them as English letters. After parsing them, every word becomes a string of elementary strokes and group of strokes. For instance,

學 (to learn) can be parsed into

or 

or 

 (word) can be parsed into 

or 

or 

In this manner the string of strokes and groups of strokes can be used as input into the computer, and the string of resulting data transmitted to the output device which rearranges the strings into original words. Dr. Nan-Hong Gau, the principal of National Chiao Tung University on April 3, 1973 declared that they have succeeded in building an input-output device that will accomplish this. It is composed of two types of key punches. The first type has 88 keys, the second kind has 648 keys with every key containing one stroke or group of strokes. Also the punches have some operation symbols. (reference 2, p-121)

Example of such operation symbols include:

1.  (to combine two strokes or group of strokes side by side)

    (wood, as an elementary group of strokes)

    (wood) +  (wood) +  (forest)

2.  (to combine two strokes or groups of strokes vertically)

    (rain, as an elementary group of strokes)

    (to say, as an elementary group of strokes)

    (rain) +  (to say) +   (cloud)

3.  (to put one stroke or group of strokes into another)

    (one, as an elementary stroke)

    (mouth, as an elementary group of strokes)

    (one) + (mouth) +   (sun)

There are several defects which will prevent the device from practical use:

1. The output device is not a printer nor a typewriter therefore it is not convenient for most output.
2. The outputs are not always of the same size and shape.
3. The operation symbols are not universal enough to combine all words.

Some researchers have considered a third method to solve the I-O problem by using phonetic-symbols to represent Chinese words. Chinese has 37 phonetic symbols which were adopted by the government in February 1913. These symbols are used to standardize the pronunciation of the words and are taught in the primary school. For years, most of the people have refused to accept the phonetic symbols as the real traditional words. (reference 2, p-122)

About 40 years ago a historian Mr. Yun-Wu Wang analyzed Chinese words and found that the four angles of a word can be numbered according to its strokes. He developed several simple rules which can be followed to assign four numbers to the four angles. Anyone can learn the rules in a few days. After learning the rules, one can write the four angle numbers as soon as he reads the word. Wang's discovery made a revolutionary improvement in dictionary compilation. In the four angle number dictionary, all words are arranged according to the four angle numbers, thus one can find a word in this dictionary much faster than the older type of dictionary. We can not use the four angle number as the code of the word, because many words have the same numbers. We also can not use the phonetic symbols as the code of the word for the same reason.

However in this paper the author will propose the combination of the phonetic symbols procedure and the four angle method to represent the word (referred to as Symbol-Number Representation) for the following reasons:

1. Only a few words have the same Symbol-Number Representation. In the 12000 words dictionary we find 40 pairs of words, each pair has the same Symbol-Number Representation. We will solve this problem in Chapter II.

2. A trained card punch operator can punch the Symbol-Number Representation on the card immediately as she reads the word without any reference.

3. There is no difficulty in making a card punch with 37 phonetic symbols, 10 digits (0-9) and other necessary special characters.

4. Electric typewriters have been built in which all words are aligned in a plain cabinet frame. The typist controls a handle with vertical and horizontal levers to search for word in the frame. After she finds the word she needs, all she needs to do is to press the levers slightly, the word will be typed in the proper place on the paper, and returned to its original position in the frame. Furthermore it is possible that high speed typewriters or printers may be manufactured in the near future, which can be connected to the computer to output the Symbol-Number Representation into real words.

The goal of this paper then is to design a computer system with an internal code of Symbol-Number Representation. In order to implement this proposal we will use an American made computer (such as the IBM 360). Thus the major work involves designing some compilers to translate

the source language of Symbol-Number Representation into a higher level language (Fortran, Cobol or Algol . . . etc.) or directly translate it into machine language. The machine language must then be translated into Symbol-Number Representation, and this information transmitted to the output device.

CHAPTER II

SYMBOL-NUMBER REPRESENTATION

The programmer usually has a choice of programming languages at his disposal. These range from machine language at the lowest level to a higher level language such as Fortran or Algol. Machine language usually involves a long and unnecessary detailed coding of the problem. In order to avoid some of the tedium of machine language, the code is usually written in assembly language or a higher order language.

Just as machine language, with its excessive detail, obscores the form of code, so assembly language, with its detail of each basic machine language operation, obscures the nature of the method used to solve the problem. To solve this, languages were developed which were intended to reflect clearly the procedure used for solution. This clarity allows the original problem to be coded more quickly and errors to be more easily removed. These languages are called procedure oriented languages. They include Fortran, Algol . . . etc.

If these higher level languages are written in Chinese, a great deal of work has to be done to get the program into memory and to begin execution at the correct points. The following steps represent a practical method to execute the Chinese procedure oriented language. All developments are based on the following two conditions:

1. The punch machine contains 37 phonetic symbols, the digits 0-9 and the necessary characters.

2. The punch operator translates the original words into Symbol-Number Representation and at the same time punches the Symbol-Number Representation on the cards.

A. Symbol-Number Representation

A Chinese word is a single sound. The pronunciation is indicated with one to three phonetic symbols. These symbols have been taught in primary school since 1928. Every Chinese speaker can write the phonetic symbols as soon as he reads the word. For example

到 (to go) uses the phonetic symbols ㄉㄠ and

若 (if) uses the phonetic symbols ㄖㄨㄛ .

Formally the symbols are written on the right side of the word.

到ㄉㄠ　　若ㄖㄨㄛ

For the purpose of coding the Symbol-Number Representation data, we will use capital letters to represent the phonetic symbols, furthermore two temporary steps are taken.

1. To make the capital letters different from the 26 English capital letters, we will prefix the capital letter to a vertical bar.

phonetic symbols	capital letters	English representation
ㄅ	B	|B
ㄆ	P	|P
ㄗ	TZ	|TZ

All the other symbols are listed in Table 2-1.

2. If two or three phonetic symbols are combined, we prefix the combined capital letters with a vertical bar.

combined phonetic symbols	combined capital letters	English representation
ㄅㄆ	BP	|BP
ㄖㄚ	RA	|RA

The pronunciation of a Chinese word is at most three phonetic symbols, but the representation of one symbol is not necessarily one letter, since some combinations contain three letters. In the coding of the Symbol-Number Representation we will use the English representation. In order to use the standard and optional length specification of variables in the system 360, we will use two rules to limit the number of characters in the English representation.

1. From Table 2-1, if two or three phonetic symbols are combined and their English representation contains three or less letters, no change is needed.
2. If two or three phonetic symbols are combined and their English representation contains more than three letters, we take the first letters as the English representation.

original words	phonetic symbols	Romanization	individual English representation	English representation	rule number
紅 (red)	ㄏㄨㄥ	H, W, ENG	|H, |W, |EG	|HWE	2
花 (flower)	ㄏㄨㄚ	H, W, A	|H, |W, |A	|HWA	1
長 (long)	ㄓㄤ	J, ANG	|J, |AG	|JAG	1

Table 2-1. Chinese phonetic symbols and their Romanization, and English representation

phonetic symbols	Romanization	English representation	phonetic symbols	Romanization	English representation
ㄅ	B	|B	ㄝ	E(E)	|EE
ㄆ	P	|P	ㄧㄚ	IA	|IA
ㄇ	M	|M	ㄟ	EI	|EI
ㄈ	F	|F	ㄠ	AU	|AU
ㄉ	D	|D	ㄡ	OU	|OU
ㄊ	T	|T	ㄢ	AN	|AN
ㄋ	N	|N	ㄣ	EN	|EN
ㄌ	L	|L	ㄤ	ANG	|ANG
ㄍ	G	|G	ㄥ	ENG	|EG
ㄎ	K	|K	ㄦ	EL	|EL
ㄏ	H	|H	ㄧ	Y or I	|Y
ㄐ	J(I)	|J(I)	ㄨ	W or U	|W
ㄑ	CH(I)	|CH(I)	ㄩ	IU	|IU
ㄒ	SH(I)	|SH(I)			
ㄓ	J	|J			
ㄔ	CH	|CH			
ㄕ	SH	|SH			
ㄖ	R	|R			
ㄗ	TZ	|TZ			
ㄘ	TS	|TS			
ㄙ	S	|S			

Table 2-1 (continued)

phonetic symbols	Romanization	English representation	phonetic symbols	Romanization	English representation
ㅏ	A	IA			
ㅗ	O	IO			
ㅓ	E	IE			

The Four Angle Number is the number assigned to a word according to its strokes. There are several rules used to assign the numbers.

0. Any point on an independent horizontal stroke is numbered 0. (An independent stroke is one that doesn't touch any other stroke).

1. Any horizontal stroke is numbered 1.

2. Any vertical or left-hand downward sloping stroke is numbered 2.

3. Any independent point or right-hand downward sloping stroke is numbered 3.

4. Any cross or cross of St. Andrew is numbered 4.

5. A vertical stroke that passes through more than two strokes is numbered 5.

6. Any square is numbered 6.

7. Any angle is numbered 7.

8. Two strokes positioned like the word 八 (eight) is numbered 8.

9. Two strokes separated by a vertical stroke like the word 小 (small, little) is numbered 9.

10. Number the left upper angle first, then the right upper angle, then the left lower angle, and finally the right lower angle.

11. If the two upper angles can only be assigned one number, the left upper angle gets the number, and the right upper angle gets 0. If the two lower angles can be assigned only one number, the left lower angle gets the number, and the right lower angle gets the number 0.

Examples:

(book)

Right upper angle numbered 5. (rule 5)
Left upper angle numbered 0. (rule 11)
Right lower angle numbered 6. (rule 6)
Left lower angle numbered 0. (rule 11)

(table, chart)

Right upper angle numbered 5. (rule 5)
Left upper angle numbered 0. (rule 11)
Right lower angle numbered 7. (rule 7)
Left lower angle numbered 3. (rule 3)

(very)

Right upper angle numbered 2. (rule 2)
Left upper angle numbered 7. (rule 7)
Right lower angle numbered 2. (rule 2)
Left lower angle numbered 3. (rule 3)

(look)

Right upper angle numbered 2. (rule 2)
Left upper angle numbered 0. (rule 11)
Right lower angle numbered 6. (rule 6)
Left lower angle numbered 0. (rule 3)

If we combine the phonetic symbols and four angle numbers to represent a word, every word has its own representation (there are exceptions which will be explained later). We write the phonetic symbols first then add a suffix to these symbols representing the four angle numbers. This is the Symbol-Number Representation. Every representation has at most seven characters (in coding, the English representations are used to represent the phonetic symbols, so some representations may have eight characters, since we added a vertical bar). For example,

original words	original Symbol-Number Representation	English representation of Symbol-Number Representation
讀 (to read)	ㄉㄨˊ 0468	|DW0468
寫 (to write)	ㄒㄧㄝˇ 3032	|SYE3032
實 (real)	ㄕˊ 3080	|SH3080
數 (number)	ㄕㄨˋ 5844	|SHW5844
老 (old)	ㄌㄠˇ 4401	|LAU4401

From a 12000 word dictionary ("The Chinese Dictionary," tenth edition, June 1, 1967 by Cultural Book Company, Taipei, Taiwan) we find 40 pairs and three triples, each of them having the same Symbol-Number Representation. We take two steps to solve this problem.

1. We select the word of one pair that is more frequently used as the first one, and use the original Symbol-Number Representation to represent it. We insert the first phonetic symbol ㄅ between the phonetic symbols and the four angle numbers of the second word.

2. We select the most frequently used word in the triple as the first word and use the original Symbol-Number Representation to represent

it. Insert the first phonetic symbol ㄅ between the phonetic symbols and the four angle numbers of the second frequently used word to represent it. Insert a second phonetic symbol ㄆ in the same place to represent the last word.

In Table 2-2 we have listed the duplicate Symbol-Number Representation pairs and triples. The phonetic symbols ㄅㄆ were added to modify the duplicate Symbol-Number Representation characters.

B. Data Representation

The most familiar method of data representation in commercial and scientific applications using computers has been binary coded decimal in which six bits are used to represent 64 alphanumeric and special characters. Several data formats have been used for processing data with the System 360 to accommodate commercial and scientific applications. An eight bit unit of information, called the byte, is the basic unit in these formats. Eight rather than six bits are used to represent a character, thus up to 256 possible characters can be represented in this Extended Binary Coded Decimal Interchange code (EBCDIC). There are 144 blank entries in the 256 possible entries. We will take advantage of the blank entries to code the phonetic symbols. Figures 2-1, 2-2, 2-3, and 2-4 show the coded characters of the System 360, and the new coded phonetic symbols.

The hole pattern of the punched card is shown at the bottom and to the right of each figure. The figures show the bit position of the coded characters. The beginning four bits are at the top and the subsequent four bits are to the left of the figure. For example, the bit

Table 2-2. Words having the same Symbol-Number Representation.

Original words	Original Symbol-Number Representation	Modified Symbol-Number Representation
鷹 (eagle)	ㄧㄥ 0022	ㄧㄥ 0022
膺 (to undertake)	ㄧㄥ 0022	ㄧㄥㄅ 0022
高 (high, tall)	ㄍㄠ 0022	ㄍㄠ 0022
膏 (paster)	ㄍㄠ 0022	ㄍㄠㄅ 0022
風 (wind)	ㄈㄥ 7721	ㄈㄥ 7721
鳳 (the male phoenix)	ㄈㄥ 7721	ㄈㄥㄅ 7721
企 (to stand on tiptoe and look for)	ㄑㄧ 8010	ㄑㄧ 8010
气 (breath)	ㄑㄧ 8010	ㄑㄧㄅ 8010
筆 (pen)	ㄅㄧ 8850	ㄅㄧ 8850
篳 (a kind of bean)	ㄅㄧ 8850	ㄅㄧㄅ 8850
縻 (to tie up)	ㄇㄧ 0029	ㄇㄧ 0029
糜 (rice gruel)	ㄇㄧ 0029	ㄇㄧㄅ 0029
麋 (the tailed deer	ㄇ 0029	ㄇㄨ 0029

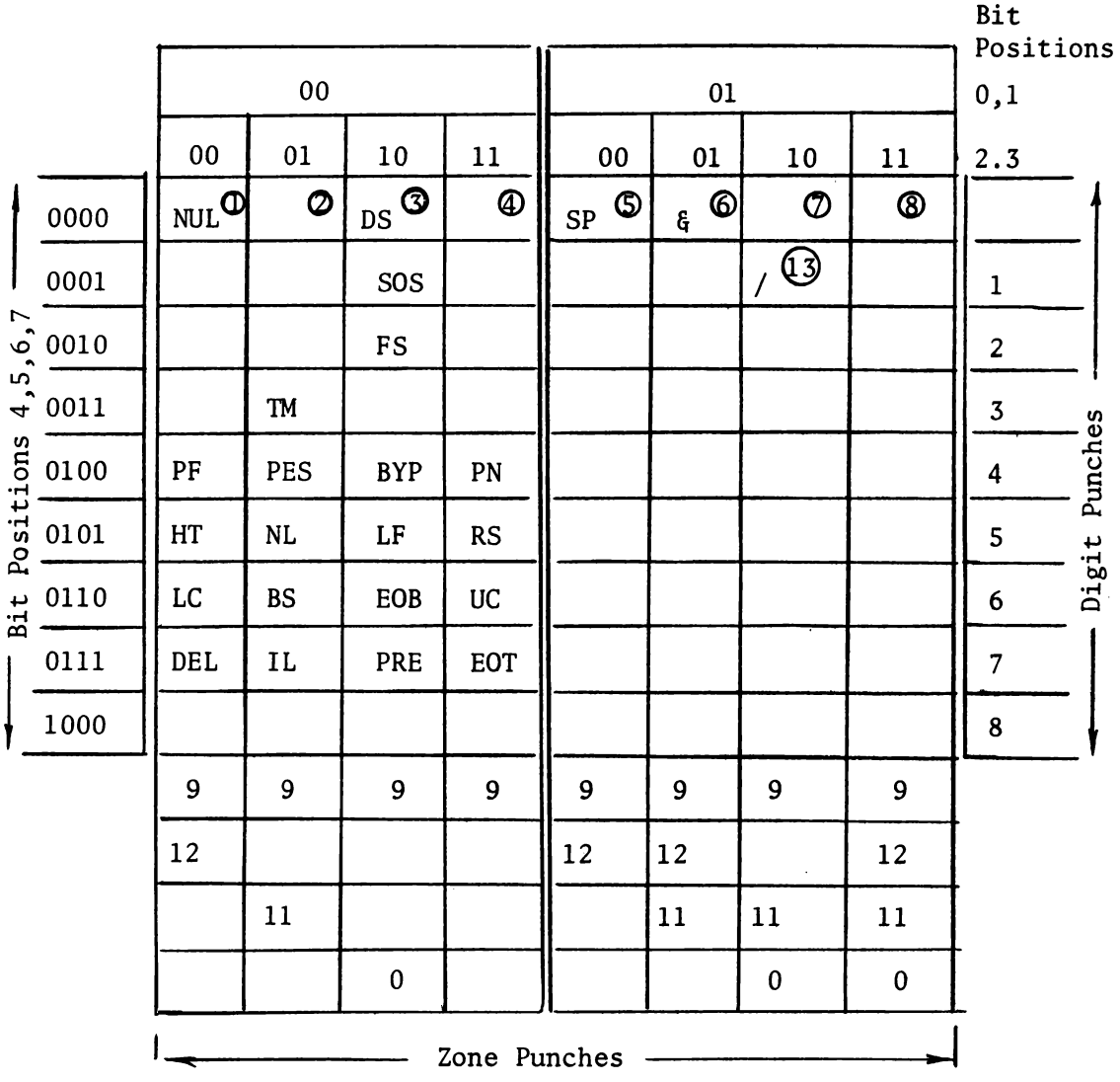


Figure 2-1. Extended binary coded decimal interchange code (a).

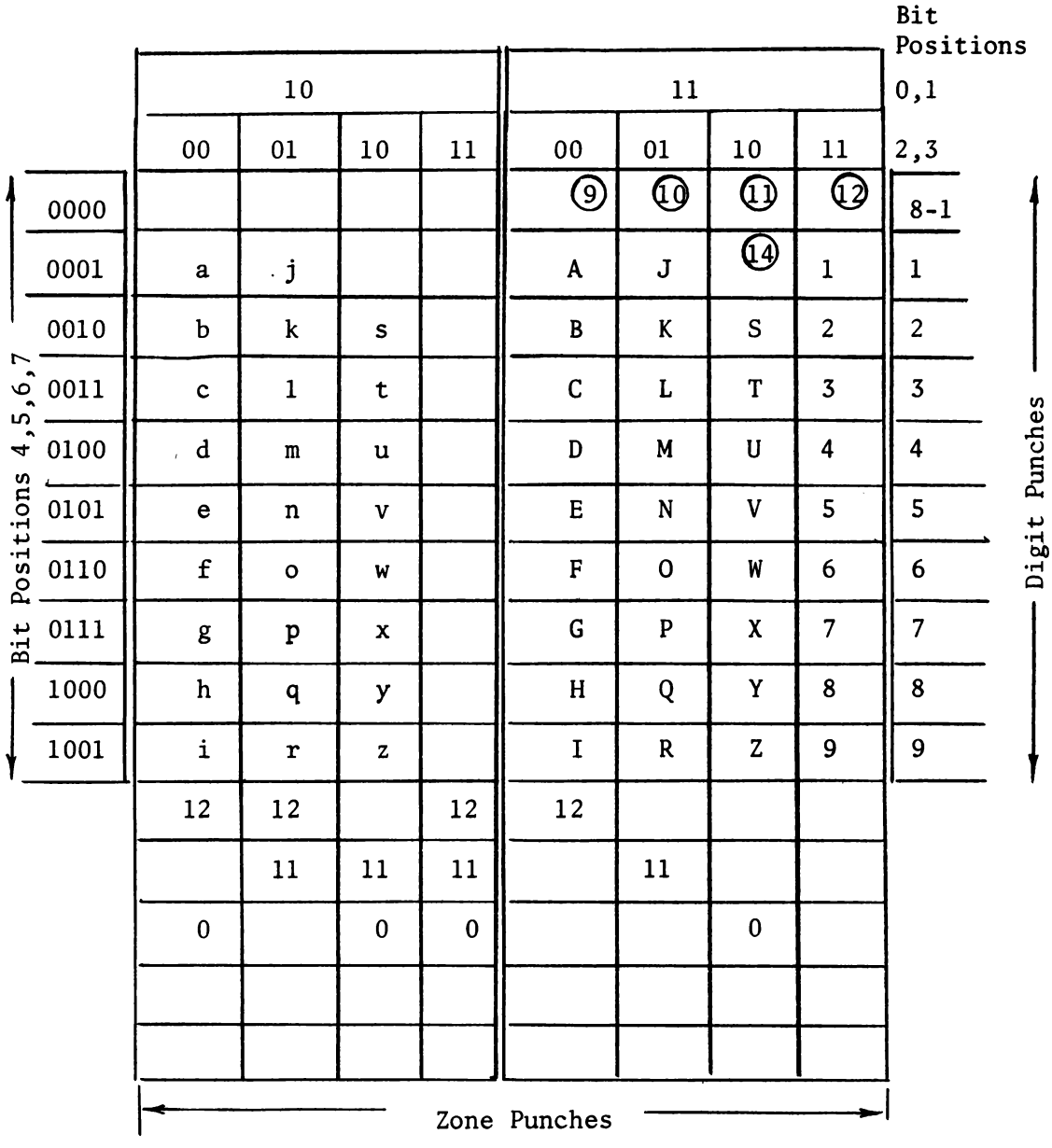


Figure 2-2. Extended binary coded decimal interchange code (b).

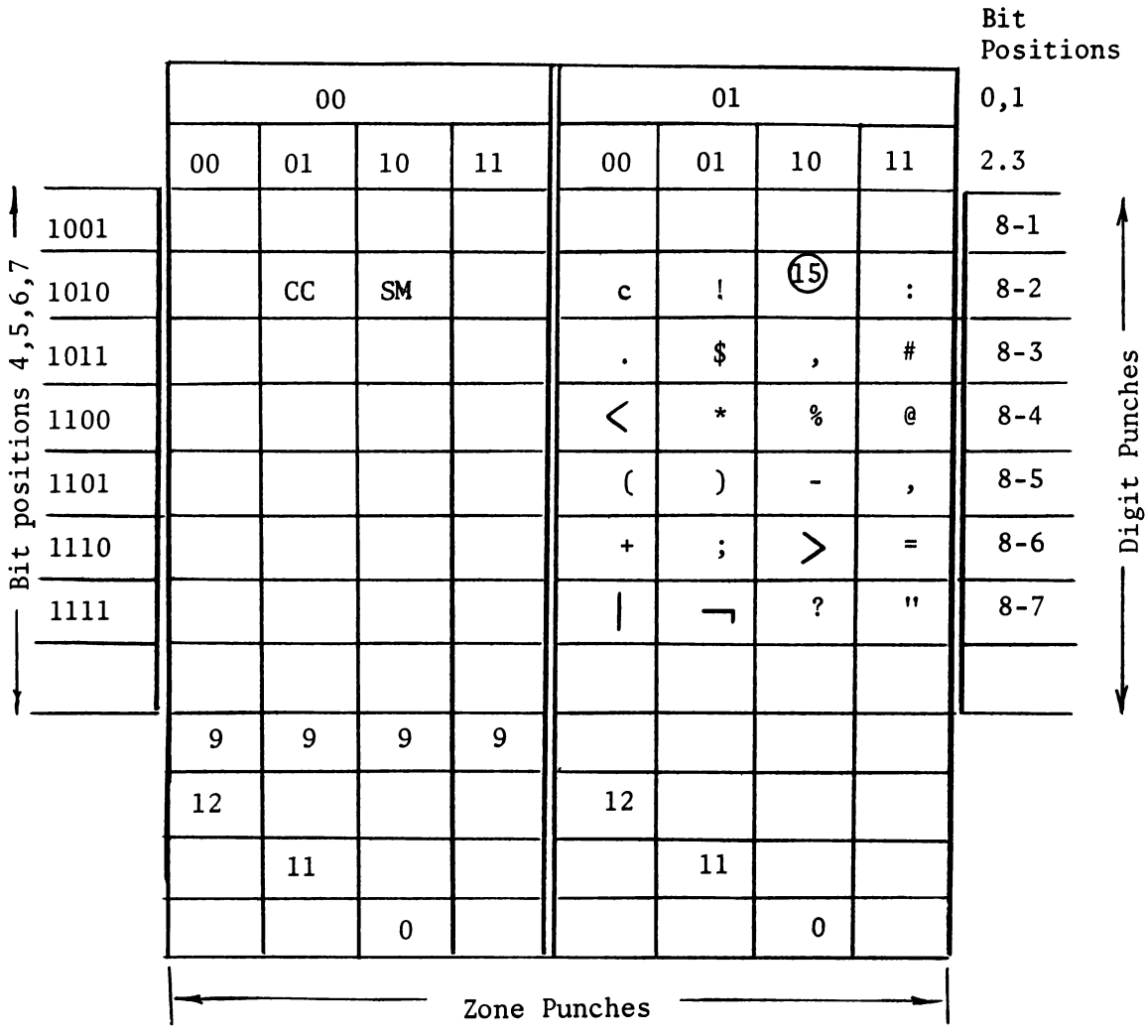


Figure 2-3. Extended binary coded decimal interchange code (c).

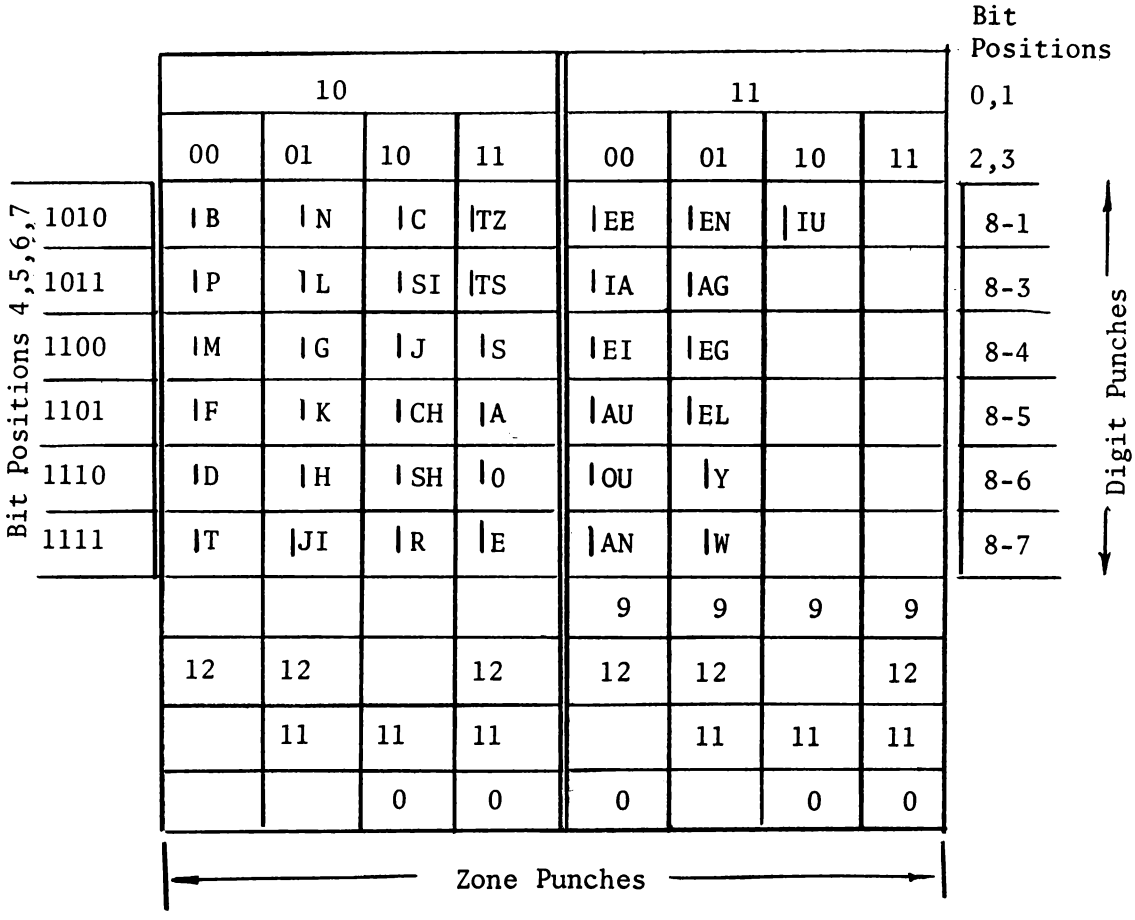


Figure 2-4. Extended binary coded Chinese phonetic symbols.

(The proper entries have been included with the English representation of the phonetic symbols)

positions of "A" are 1100(from top)0001(from left), and for "M" are 1101(from top)0100(from left). The character "A" in the punched card is represented by a hole in zone position 12 and a hole in digit position 1.

For an example in system 360, to translate the name John Dor into EBCDIC, the result is

11010001	10010110	10001000	10010101	11000100
J	o	h	n	D
10010110	10011001			
o	r			

in Symbol-Number Representation, translated the word 讀 (to read) DW0468 into EBCDIC, the result is

10011110	11011111	11110000	11110100	11110110
❙D	❙W	0	4	6
11111000				
8				

The control characters on the left hand side of Figure 2-1 such as FS, PF, PN . . . etc. are the characters which can perform proper intern internal control operation such as line feed, backspace . . . etc. The explanation of their meaning is given in Table 2-3.

Exceptions to the tabular representation of hole patterns to specify a binary bit pattern, a control character, or a graphic character are identified by numbers circles in the table. The proper hole patterns for these characters are shown in Table 2-4. For example, from Figure 2-1 the character "&" is marked with a circled 6. From Table 2-4

Table 2-3. Control characters.

NUL	Null	SOS	Start of Significance
PF	Punch Off	FS	Field Separator
RT	Horizontal Tab	BYP	Bypass
LC	Lower Case	LF	Line Feed
DEL	Delete	EOB	End of Block
TM	Tape Mark	PRE	Prefix
RES	Restore	SM	Set Mode
NL	New Line	PN	Punch On
BS	Backspace	RS	Reader Stop
IL	Idle	UC	Upper Case
CC	Cursor Control	EOT	End of Transmission
DS	Digit Select	SP	Space

Table 2-4. Exceptions to normal tabular representations

1	12-9-8-1	9	12-0
2	12-11-9-8-1	10	11-0
3	11-0-9-8-1	11	0-8-2
4	12-11-0-9-8-1	12	0
5	No Punches	13	0-1
6	12	14	11-0-9-1
7	11	15	12-11
8	12-11-0		

(6) is 12, this means that the hole punched for the character "&" is at 12 instead of 12-11-9. The character '/' is marked with a circled 13. From Table 2-4, (13) is 0-1 and the hole pattern of '/' is 0-1 instead of 9-11-0-1.

Table 2-5 lists the special graphic characters used and Table 2-6 consists of examples which show how one can use the figures and tables to represent different characters.

Using the figures and tables, we can see that the new coded phonetic symbols do not conflict with the other coded members. Thus we can use either upper cases or lower case coded English letters plus all the special symbols. In the System 360, we need not change any of the specifications, since all we need to do is to use these new coded entries for the Symbol-Number Representation.

Table 2-5. Special graphic characters.

c	Cent Sign	-	Minus, Hyphen
.	Period, Decimal Point	/	Slash
<	Less-than Sign	,	Comma
(	Left Parenthesis	%	Percent
+	Plus Sign	_	Underscore
|	Vertical Bar, Logical OR	>	Greater-than Sign
&	Ampersand	?	Question Mark
!	Exclamation Point	:	Colon
$	Dollar Sign	#	Number Sign
*	Asterisk	@	At Sign
)	Right Parenthesis	'	Prime, Apostrophe
;	Semicolon	=	Equal Sign
¬	Logical Not	"Quotation Mark	

Table 2-6. Representation of different types of characters.

Example	Type	Bit pattern Bit positions 0 1 2 3 4 5 6 7	Hole Pattern Zone punches | Digit punches
LF	Control Character	0 0 1 0 0 1 0 1	9-0-5
PF	Control Character	0 0 0 0 0 1 0 0	12-9-4
%	Special Graphic	0 1 1 0 1 1 0 0	0-8-4
/	Special Graphic	0 1 1 0 0 0 0 1	0-1
-	Special Graphic	0 1 1 0 0 0 0 0	11
⑭	Control Character (function not yet assigned)	1 1 1 0 0 0 0 1	11-0-9-1
R	Upper Case	1 1 0 1 1 0 0 1	11-9
a	Lower Case	1 0 0 0 0 0 0 1	12-0-1
④	Control Character (function not yet assigned)	0 0 1 1 0 0 0 0	12-11-0-9-8-1
ㄇM	Chinese Phonetic Symbol	1 0 0 0 1 1 0 0	12-8-4
ㄠAU	Chinese Phonetic Symbol	1 1 0 0 1 1 0 1	12-9-0-8-5

CHAPTER III

THE SCANNER AND RECOGNIZER

A. Scanner

After the input/output section processes the source program and other data, the source program has to be analyzed and decomposed into its basic parts and then translated into equivalent object parts. The scanner performs a single lexical analysis and constructs the source program symbols (key words, identifiers . . . etc.). The scanner then provides the syntax analyzer a table containing the source program in an internal representation form.

Before we demonstrate how the scanner operates (Figure 3-1), we need to define the symbols which will be used in the symbol-Number Representation language, since these are the symbols which the scanner must build.

1. The English representation of the phonetic symbols are treated as single characters, such as |B, |D, |K . . . etc.
2. For the purpose of simplification, we will define the following symbols,

   SC means "special characters"

   FAN means "four angle numbers", such as 1987, 9852

   ..... etc.

   EPS means "English representation of phonetics

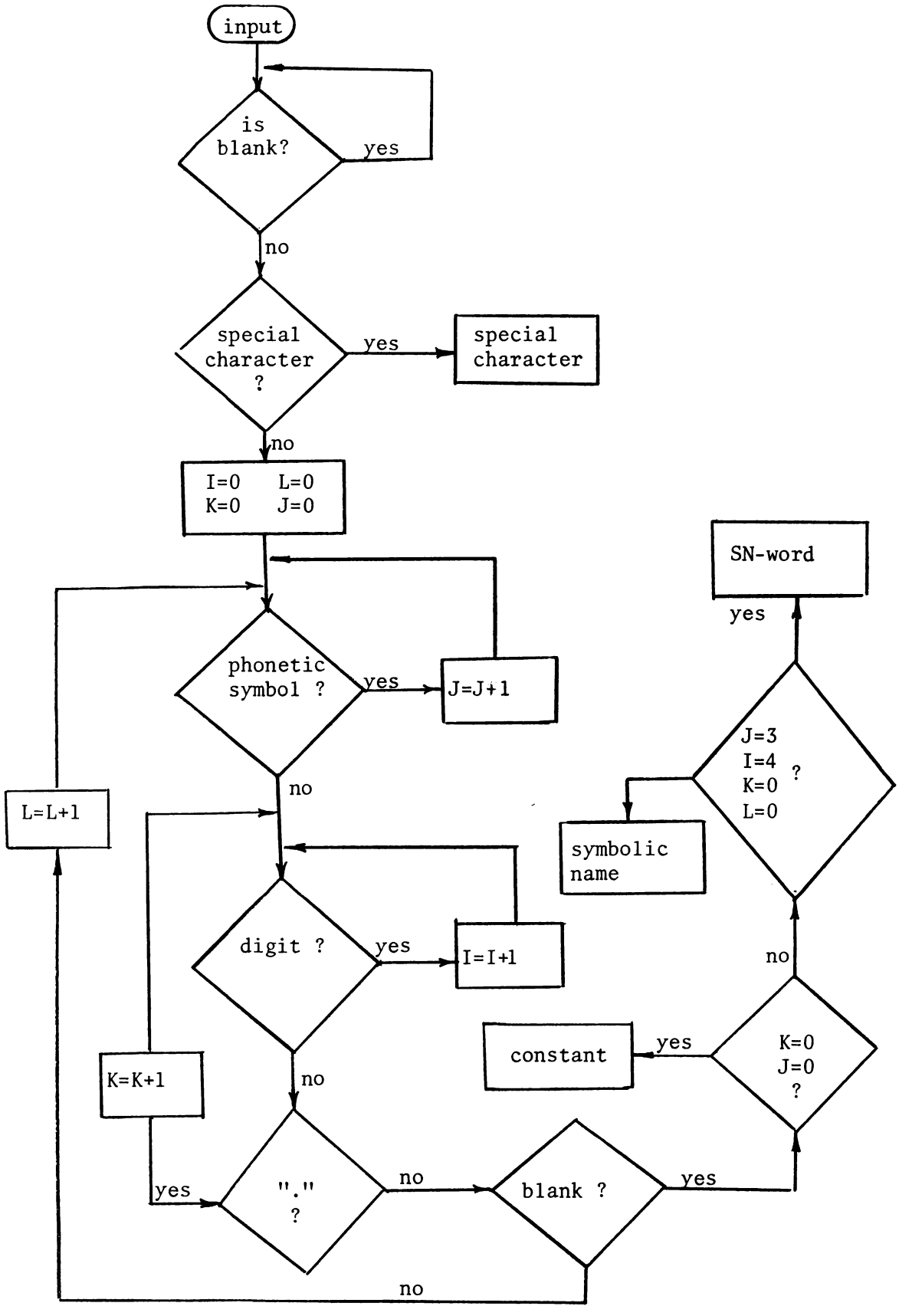


Figure 3-1. Flow diagram of the scanner.

Sn-word means "a Chinese word represented with Symbol-Number Representation".

SNA means "symbol name" which is a string of phonetic symbols and digits, the first character must be a phonetic symbol, or a string of letters and digits, the first character must be a letter.

VN means "variable name" which is a "symbol name", but the number of the characters must not be greater than six.

I means "identifier".

C means "constant".

The following Backus form will be followed for our scanner.

<SN> ::=* | & | # . . . . |/

<EPS> ..=|B | |K . . . . | | TZ

<digit> ::=0 | 1 | 2 . . . |9

<letter> ::=A | B . . . . | Z

<SN> ::= <EPS> (<EPS> | <digit>)^{n} $n \leq 7$

<I> ::= VN

<FAN> ::= <digit>^{4}

<SN-word> ::= <EPS>^{n} <FAN> $1 \leq n \leq 3$

B. Recognizer

After the scanner transforms the input characters into symbols, integers, special characters an SN-words, the next step is to determine if the input is a legitimate string and determine what structure it has. That is we want to construct a syntax recognizer. The two primary methods used for recognizing the structure of the input string are the bottom-up and top-down approaches. There are many types of bottom-up methods. Each method involves the process of scanning the input string and combining characters to form more complex strings. The operator precedence method relies on the fact that certain of the characters or symbols of the input string can be recognized as operators and that the other elements are recognized as operands. A precedence relationship, or hierarchy, between the operators enables the program to decide which operator should be compiled first. We will use this method to recognize our arithmetic sentential forms. This technique uses the following three steps until the sentence has been reduced to the distinguished symbol.

a. Find the handle x (or some variation of it),

b. Find the rule U ::= x with x as a right part,

c. Reduce x to U, creating one branch of the syntax tree.

The following principle will illustrate how to use the three steps a, b, and c.

1. If P is a set of productions containing one symbol S which appears on the left of ":-" and a set of terminals I or C appearing on the right side of ":-" , P is called Phrase Structure Grammar.

If there is no production of P in the form U := xEFy where E and F are nonterminals, then no sentential form contains two adjacent nonterminals. Hereafter, we use the terminals as operators and the nonterminals as operands.

2. there are three relations between two terminals I_1 and I_2.

   (a) $I_1 = I_2$ if there is a production T := xI_1I_2y or T := xI_1EI_2y, where T, E are nonterminals, x,y are strings of terminals.

   (b) $I_1 > I_2$ if there is a production T := xEI_2y and E := z, where T, E are nonterminals, x, y, z are strings of terminals. I_1 is the rightmost terminal of z.

   (c) $I_1 < I_2$ if there is a production T := xI_1Ey and E := z, where T, E are nonterminals, x, y, z are strings of terminals. I_2 is the leftmost terminal of z.

3. An operator precedence grammar is an operator grammar for which no more than one of the above three relations holds between any ordered pair I_1, I_2 of terminals.

4. A prime phrase is a phrase of a sentential form which contains at least one terminal, but no prime phrase other than itself. The sentential form E * E (I + F) contains the prime phrase E * E and I. Every other prime phrase either consists of a nonterminal or contains one of the prime phrases.

5. We will write a general sentential form, enclosed between two delimiters # and # as

   # N_1 I_1 N_2 I_2 ..... N_n I_n $N_{(n=1)}$ #

   where I_i are terminals, N_i are the nonterminals. A useful theorem will be used to recognize the prime phrase in the sentential form. Theorem: The leftmost prime phrase of a sentential form of an operator precedence grammar is the leftmost substring

   N_j I_j . . . . N_i I_i $N_{(i+1)}$ such that $I_{(j-1)} < I_i$,

   $I_j = I_{(j+1)}$ ..... $I_{(i-1)} = I_i$, $I_i > I_{(i+1)}$

To illustrate the above principle, we will describe the recognition process for arithmetic sentential forms. Usually we describe the arithmetic expression with the grammars,

⟨factor⟩ := ⟨identifier⟩
:= ⟨constant⟩
:= ⟨(expression)⟩

⟨term⟩ := ⟨factor⟩
:= ⟨term⟩ * ⟨factor⟩
:= ⟨term⟩ / ⟨factor⟩

⟨expression⟩ := ⟨term⟩
:= ⟨expression⟩ + ⟨term⟩
:= ⟨expression⟩ - ⟨term⟩

For simplification, we write

F for Factor (nonterminal)

T for Term (nonterminal)

C for Constant (terminal)

I for Identifier (terminal)

E for Expression (nonterminal)

then, the grammar can be written as,

1. F ::= I
2. F ::= C
3. F ::= (E)
4. T ::= F
5. T ::= T * F
6. T ::= T/F
7. E ::= T
8. E ::= E + T
9. E ::= E - T

From rules 1 to 9 we can construct Table 3-1, where F, E, T are nonterminals, C, I, *, /, +, - are the terminals.

From Table 3-1 and the rules 1, 2, 3, listed earlier, a precedence matrix can be constructed as shown in Figure 3-2.

At each step of the precedence operation process (shown in Figure 3-3) a number of terminals will be reduced to nonterminals. Furthermore, the process must be completed in a finite number of steps with the result that S (m) = # N #. For example, to recognize the prime phrase T(0) = # I * (I + I) #, the first step will be reduced to T(1) = # F * (I + I) #, then T(2) = # F * (F + I) #, and so on. In Table 3-2 T(k) is the input sentential form, S(j) contains the operators and operands of T(k) being processed, and Q(n) is the stack

Table 3-1. Leftmost and rightmost terminals of F, T, E

nonterminals	leftmost terminals	rightmost terminals
F	I, C, (	I, C,)
T	I, C, (, *, /	I, C,), *, /
E	I, C, (, *, /, +, -	I, C,), *, /, +, -

	C	(	*	/	+	-	)	I	#
#	<	<	<	<	<	<	<	<	=
(	<	<	<	<	<	<	=	<	>
*	<	<	>	>	>	>	>	<	>
+	<	<	<	<	>	>	>	<	>
/	<	<	>	>	>	>	>	<	>
-	<	<	<	<	>	>	>	<	>
)	|	|	>	>	>	>	>	|	>
C	|	|	>	>	>	>	>	|	>
I	|	|	>	>	>	>	>	|	>

Figure 3-2. Precedence matrix.

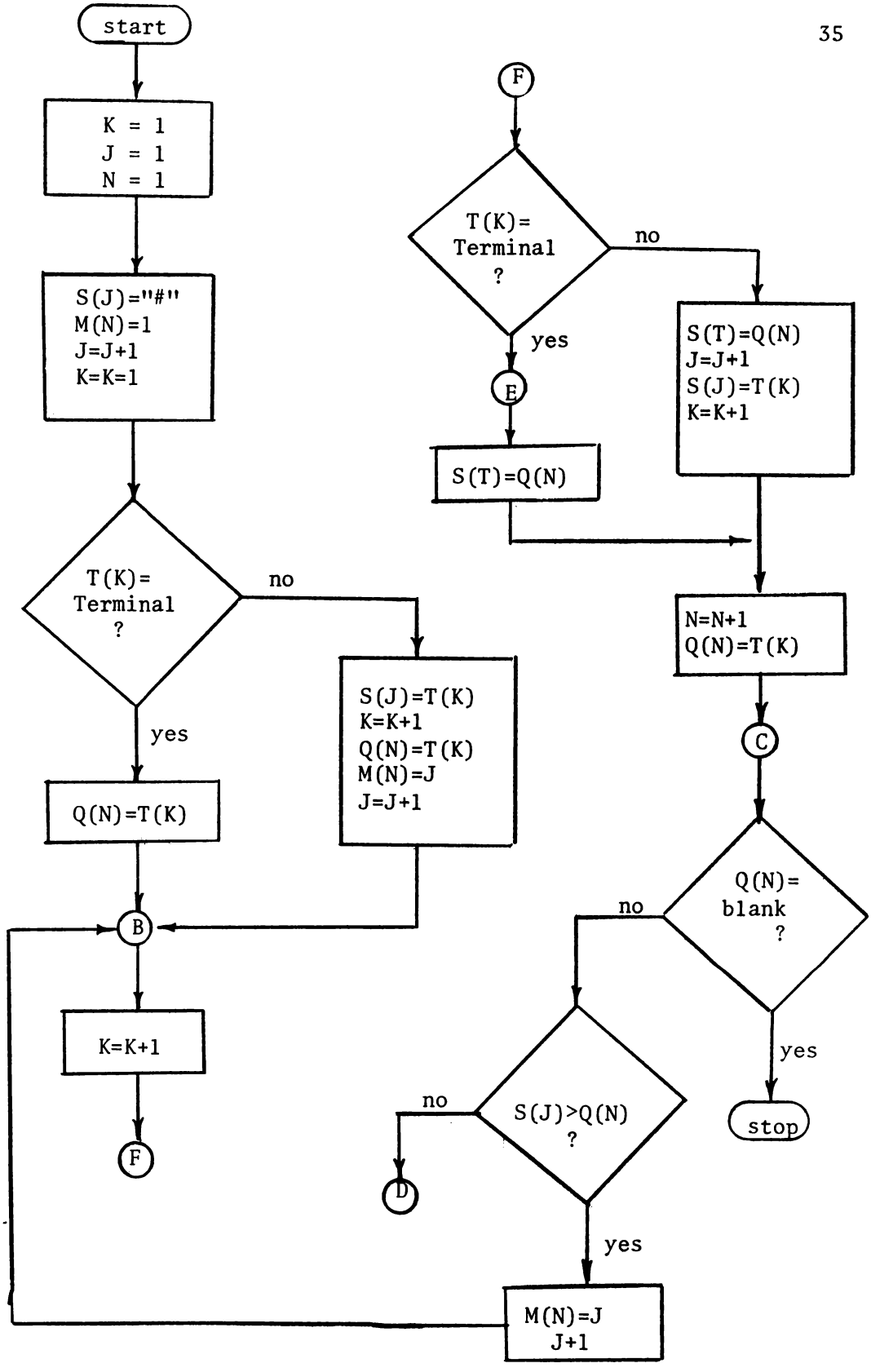


Figure 3-3. The flow diagram of operator procedure recognizer.

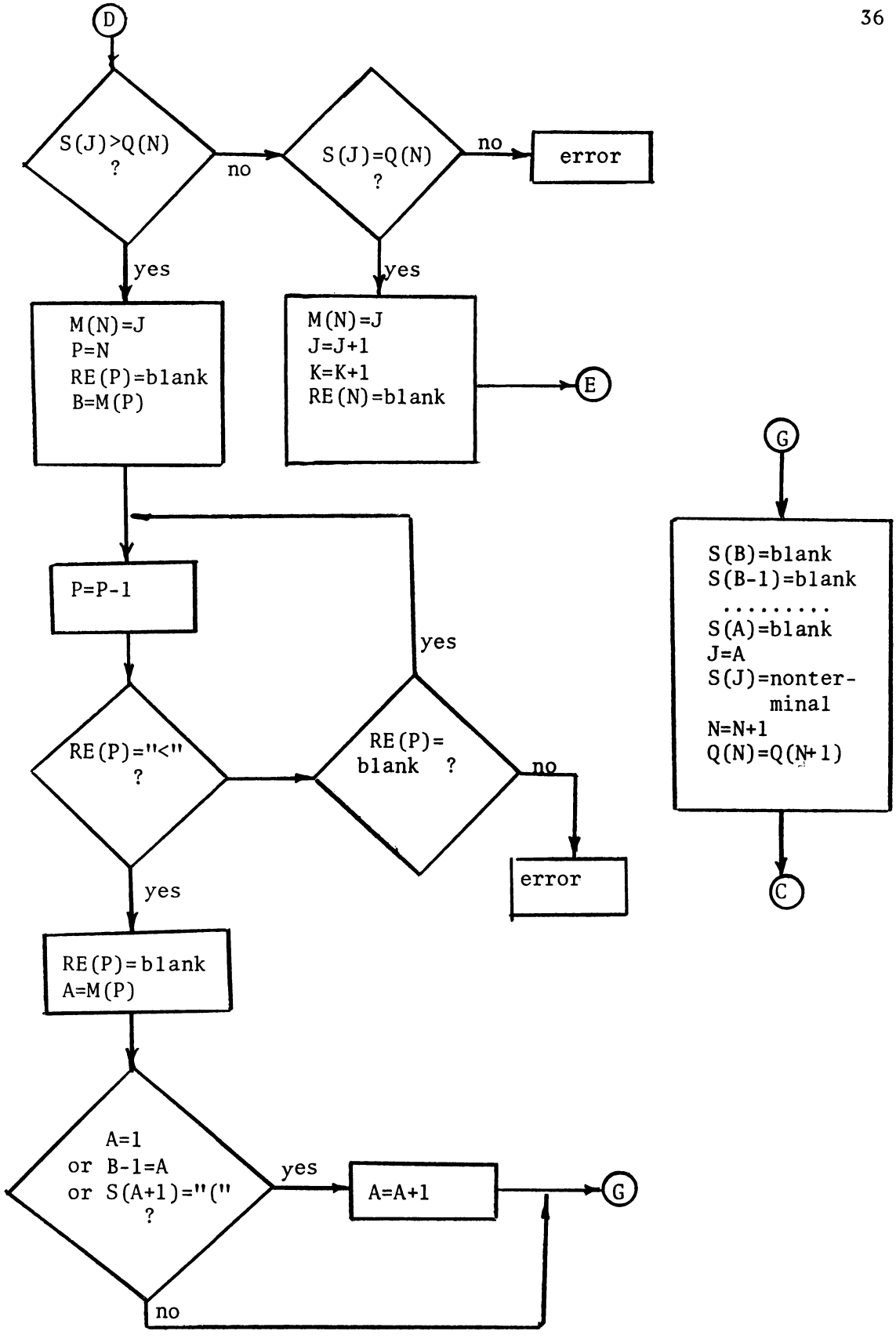


Figure 3-3 (continued)

Table 3-2. Recognition processes of # (I + I) * I #

step n	M(n) = j	S(j)	RE(n) (precedence relation)	Q(n)	T(k)
1	1	#	<	(	I + I) * I #
2	2	# (	<	I	+ I) * I #
3	3	# (I	>	+	I) * I #
4	3	# (N	<	+	I) * I #
5	4	# (N +	<	I	) * I #
6	4	# (N + I	>	)	* I #
7	4	# (N + N	>	)	* I #
8	3	# (N	=	)	* I #
9	4	# (N)	>	*	I #
10	2	# N	<	*	I #
11	3	# N *	<	I	#
12	4	# N * I	>	#	
13	4	# N * N	>	#	
14	2	# N	=	#	
15	3	# N #			
16	3	stop			

containing the leftmost operator of T(k) for comparison with the rightmost operator of S(j). The processes are:

1. Put the left delimiter of T(k) in S(1), if T(2) is a terminal, transfer the terminal into Q(1),
2. If T(2) is a nonterminal, put it into S(2), then T(3) must be a terminal, transfer the terminal into Q(1),
3. Comparing S(j) and Q(n) for every step n, we can determine the precedence relation from the precedence relation matrix,
4. If S(j) is a nonterminal, S(j-1) must be a terminal, then compare S(j-1) instead of S(j) with Q(n).
5. At step n_1 the precedence relation is <, when we get the precedence relation > at step n_2, the operators and operands transferred into S(j) can be reduced with the grammars 1. to 9.

CHAPTER IV

INDIRECT TRANSLATION

Language translation takes a source language into an object language. In our case the source language is the Symbol-Number Representation programming language, and the object language is a programming language such as Fortran, Algol . . . etc.

For language translation purposes, the definition of the source language must be given in terms of the object language. Much of the burden of the success of the translation process is placed on the accuracy and completeness with which these syntactical and semantic descriptions are made. The translator is a program, written in a meta-language which operates only on syntactic and semantic structures.

Let S(L) be the source language, and O(L) be the object language. For our translation method, we desire the definition of source language in terms of the object language, that is S(L) = S(O(L)), where O(L) is given by the computer manufacture's programming manual.

The translator can be considered as the transformation T such that T(S(L)) = O(L), with the property that if P(S) is a program written in the source language, T(P(S)) = P(O(L)), where P(O(L)) is the translated program in terms of the object language. Thus the translator operates by looking at the successive symbols of the program written in the source language, and transforms them into symbols of the object language, according to the syntax and semantics of the description.

As an example, we will use the sentential form, 3|GE479b (2|I4, 2(|D5)). This is a Format statement in Chinese and our scanner can parse it as <integer> <SN-word> <special character> . . . . and so on. Tedious work has to be done to translate it. First we need to decide if it is legal and what kind of statement it is. Then we translate the symbols into the proper object language code. For this input sentential form, the syntax rules are given in Table 4-1. Now we employ the following successive steps.

1. Take the first symbol of the string and look on the right hand side of the syntax rules to identify its syntactical form. The first rule indicates it is an <integer>. Our next step is to determime if it can be developed a part of some higher order syntax form.
2. The second symbol is a <SN-word> which is the <keyword>, from which we can identify what kind of statement we are processing.
3. The third symbol is a <special character>. Using rule 6, in Table 4-1, we find <special character>, but the symbol before it is a <format code>, so no higher order rules could be formed.
4. The fourth symbol is an integer, the fifth symbol is a <special phonetic symbol>, and the sixth symbol is an <integer>. From rule 6, these three symbols are combined into the <format code>.

Table 4-1. Syntax rules for a format statement

1. <integer> ::= <digit>^{n}

2. <special character> ::= (|) | , | '

3. <character> ::= <letter> | <phonetic symbol> | <digit>
| <special character>

4. <special phonetic symbol> ::= |H | |J | |E | |TZ | |D | |G
|L | |F | |A

5. <keyword> ::= <SN-word>

6. <format code> ::= <integer> <special phonetic symbol> <integer>
| <special phonetic symbol> <integer>
| <special character> <character>^{n} <special character>
| <integer> <special phonetic symbol>
| <integer> <special phonetic symbol> <character>^{n}
| <integer> <special phonetic symbol> <integer>
<special character> <integer>

7. <format statement> ::= <integer> <keyword>
<special character> <format code> <special character>
<format code> <special character> <format code>
..... <special character>

5. The seventh symbol is a <special character>. Only rule 6 has a <special character>, and the symbol before it is a <format code>. However the symbol after it is not an <integer> so the symbol can not be combined into any higher order rules.
6. The eighth symbol is an <integer>, and the ninth symbol is a <special character> (from rule 6). The tenth and eleventh symbols are combined into <format code>, and the twelfth symbol is a <special character>, so these symbols are combined into <format code>.
7. From rule 7, the last symbol is a <special character>, the combined symbol before it is the <format code>, the symbol before the <format code> is a <special character>. . . . and so on.
8. Every symbol of the input string coincides with the symbol of rule 7. The input string is recognized as a <format statement>. Each rule of Table 4-1 is one line, so the line number is taken as the rule number (or level number) of the corresponding syntactical entity of that line. The possible forms listed on the right of "::=" are called "alternative terms". The entities that make up an alternative term are called "components", such as

<format code> ::= <integer> <special character>
| <special phonetic symbol> <integer>

There are two alternative terms ⟨integer⟩ ⟨special character⟩ and ⟨special phonetic symbol⟩ ⟨integer⟩. The vertical bar separates the various alternatives.

The components can be expressed directly as their respective level numbers. The source code being translated is said to be composed of a list of symbols, where a symbol can be a literal or a rule number. Each component of the rule has an associated semantic part, which is the object language symbols or codes.

In Chapter III we noted that the phonetic symbols and the letters do not conflict, so we can use them at the same time. But for the purpose of translation, we must translate the proper symbols into correct and legal object language symbols or codes.

For rule 4 in Table 4-1, page 41, the special phonetic symbols are |H, |J, |E, |TZ, |D, |T, |G, |L, |S, |F, |A . The corresponding descriptions (Format code) are given as H, I, E, Z, D, T, G, L, X, F, A in Fortran. In rule 5, the semantic part is equivalent to FORMAT in Fortran. Let us take (E_n) to stand for the translated symbols (or object language codes) of the corresponding grammatical form, where n is an integer indicating the order of the components from left to right in one rule. For example,

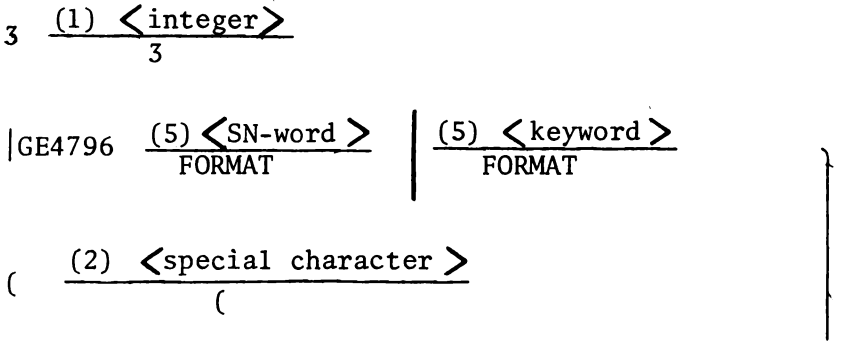
⟨format code⟩ ::= ⟨special phonetic symbol⟩ (E_1)
⟨integer⟩ (E_2)

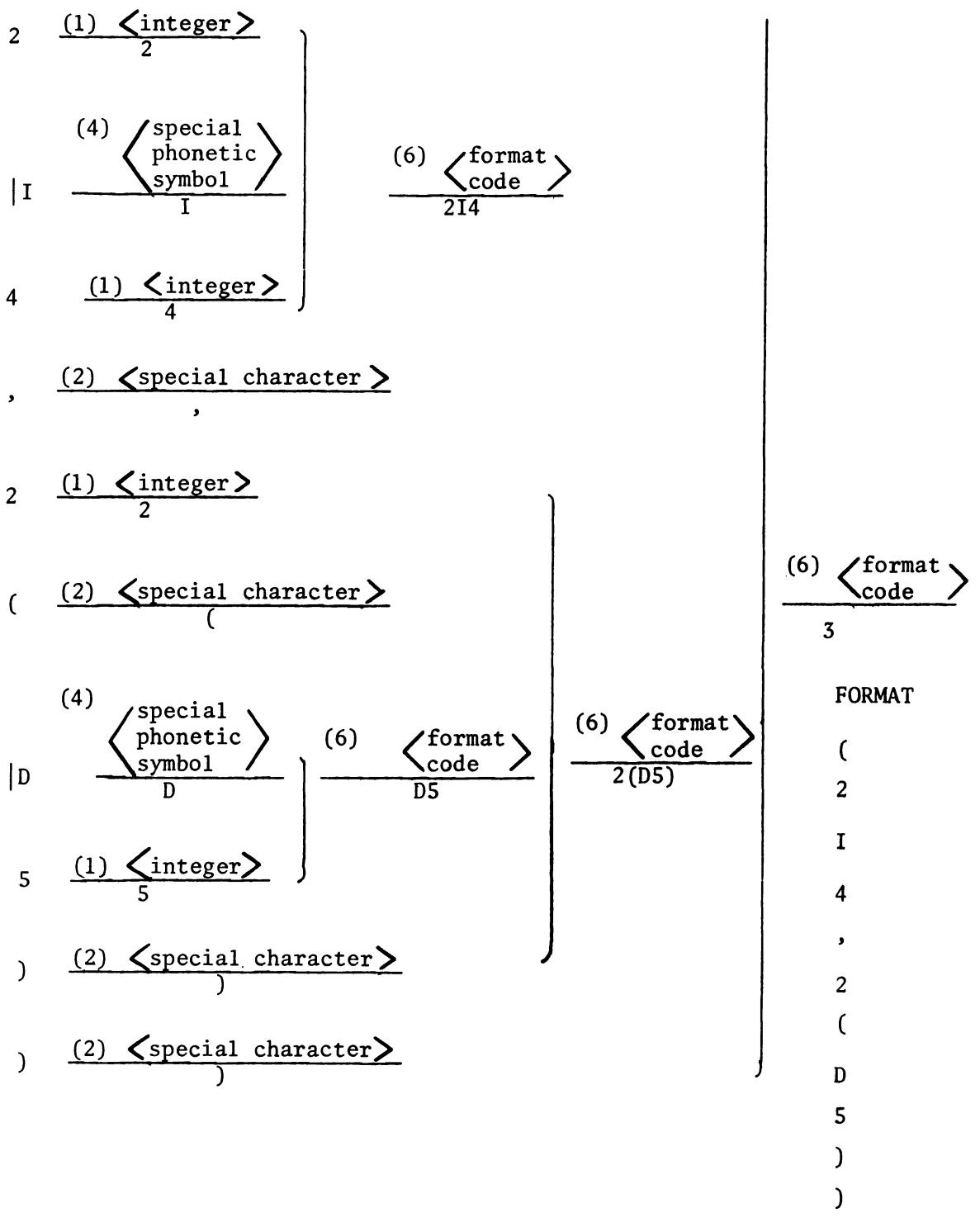
The analysis process proceeds in the manner described above. Several rules invoked during the translation include;

1. Some special phonetic symbols used in the Symbol-Number Representation statement, must be replaced with their corresponding letters.
2. Number, special characters, and variables written in symbol-digit or letter-digit form should not be changed.
3. All SN-word must be replaced with their semantic description.
4. All external functions and intrinsic functions that will be supplied by the compiler must be kept with the original symbolic name, such as, Trigonometric Sine is SIN

   Exponential is EXP ... etc.

Using the rules in Table 4-1, page 41, we can illustrate the translation and recognition processes occurring at the same time by translating a string such as 3 |GE4796 (2 |I4, 2(|D5)). We place the semantic interpretation just beneath the syntactic component.




2
(1) <integer>
2
|I
(4) <special phonetic symbol>
I
4
(1) <integer>
4
(6) <format code>
2I4
,
(2) <special character>
,
2
(1) <integer>
2
(
(2) <special character>
(
|D
(4) <special phonetic symbol>
D
5
(1) <integer>
5
(6) <format code>
D5
)
(2) <special character>
)
)
(2) <special character>
)
(6) <format code>
2(D5)
(6) <format code>
3
FORMAT
(
2
I
4
,
2
(
D
5
)
)

The use of level numbers makes this analysis more efficient. We think of each symbol or combination of symbols in the sentence as being successively replaced by one with a higher level number in accordance with the syntax rule, in an attempt to obtain the highest level number which will replace the entire sentence. Each time replacement is made, the corresponding semantic descriptions, or translations are aligned, as illustrated in the example.

This illustration is intended to show the possibility of automatic translation from the Symbol-Number Representation language to some other higher level language. However it is known that this method is not sufficient for some higher order translation processes, such as natural language translation.

The translator itself manipulates a list and its sublists. The list is the source language code list and the sublists are the translated fragments. The translator carries out a relatively simple recursive processing of the syntax and the symbols. It uses the operation of the comparison of a symbol of the source language and a component of the syntactic rule, and the selection of an adjacent symbol or component for processing. Each time a higher level number is found, the higher rule replaces the component in the source code list, and the associated translated object code is generated. The following notation is used to describe the translator shown in Figure 4-1.

n--indicates the symbol of the code being translated from left to right.

N--indicates the lines of the syntax description, from top to bottom.

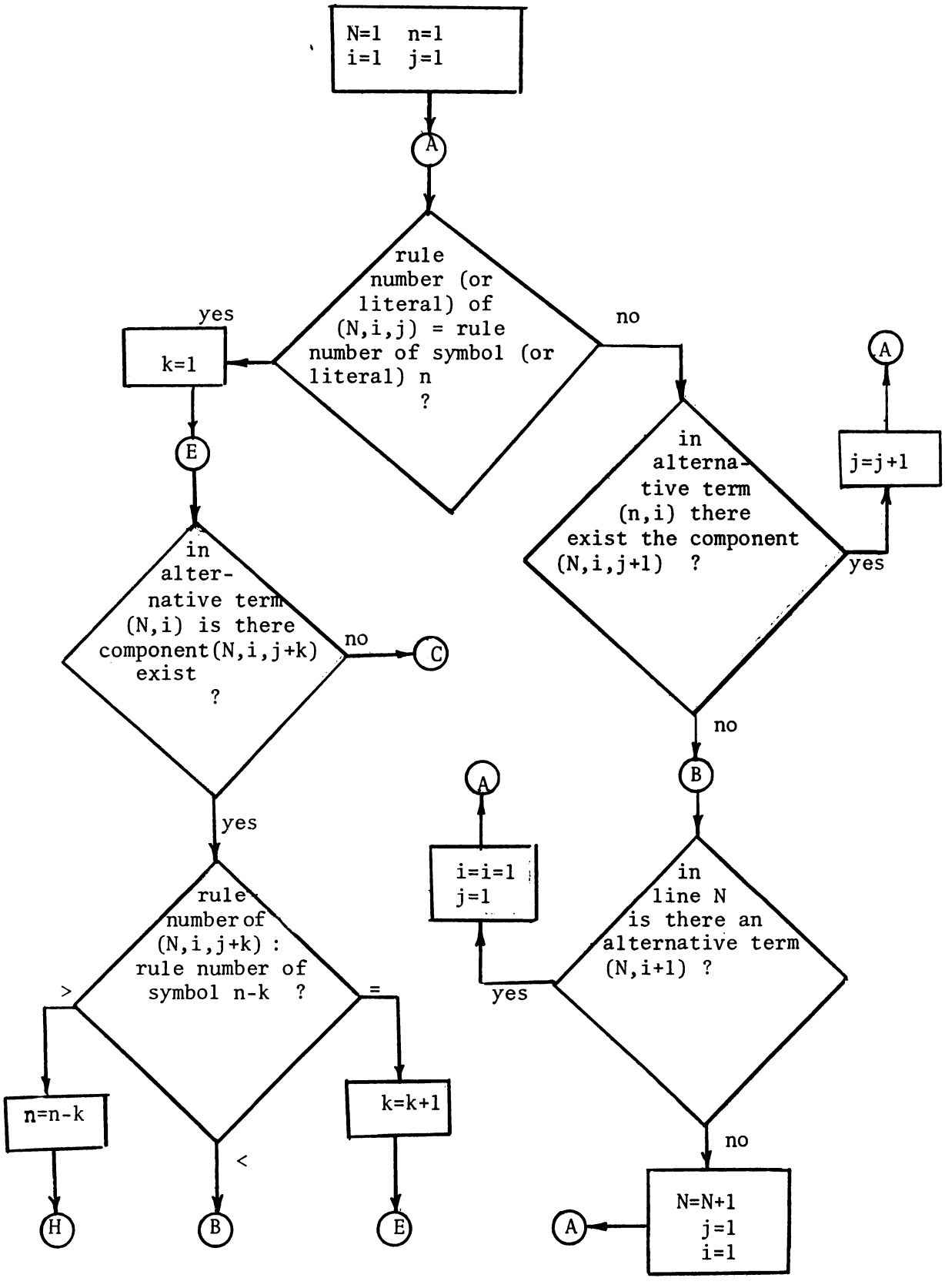


Figure 4-1. Flow chart of translation operation.

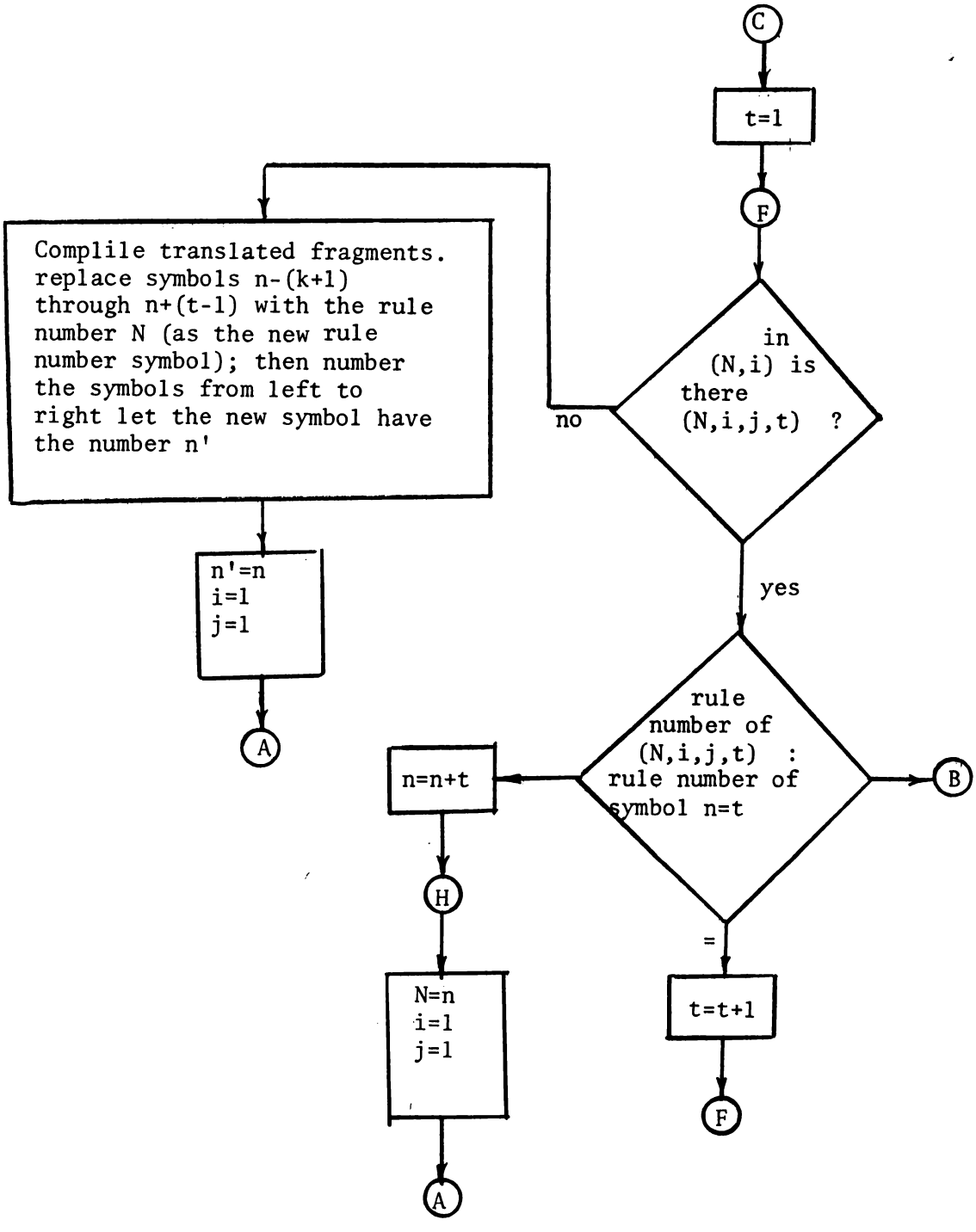


Figure 4-1 (continued)

i--indicates the alternative terms of a line, from right to left, hence N, i designates a particular alternative term.

j--indicates the components of an alternative term, from right to the left, hence N, i, j designates a particular component.

The key steps in the procedure outlined in Figure 4-1, pages 47 and 48, are,

1. The initial key symbol is the left most symbol of the source sentence (or source code) to be translated. The first component under consideration is the rightmost component of the rightmost alternative term of the syntax line.
2. Search through the successive components for one syntax line that will match the key symbol, the search proceeding from right to left and from the top line down. (The component with which the match is made is called the key component.) A syntactical replacement can be made only when all the components of an alternative term completely match a set of adjacent symbols of the source code.
3. This step systematically performs the necessary comparisons, first to the left, then to the right of the key symbol and of the key components. In general comparisons around the key symbol in both directions are required when there are three or more components in an alternative term.

When the level numbers are compared, three cases may result,

a. First, suppose every component (level number) in the alternative term of the key component is equal to its corresponding symbols (level number) in the set about the key symbol in the source code, then a higher level number replaces the set of corresponding source code symbols. The previous translation fragment associated with all those symbols are combined, and the search of step 2 is restarted at the same line. This is accomplished in step 4.

b. Second, suppose that during comparisons in step 3 a component level number is less than the corresponding symbol level number, then this certainly could never be developed so as to meet such specifications. Hence no fit can be made in that alternative term, and the search conditions as in step 2.

c. Third, suppose during the comparison of step 3, a component level number is greater than a corresponding symbol level number could be developed to match the higher component level number. This indicates that perhaps the symbol level number could be developed to match the higher component level number. Hence the procedure of step 5 is followed, and this symbol is chosen as the new key symbol. The search of step 2 is restarted on the line having the same level number as this new key symbol.

If the syntax is properly formulated and the sentence, or source code is properly formed, then the process should end with the entire source code having been replaced by the highest level number, and the translation having been completed.

CHAPTER V

DIRECT TRANSLATION

When the source language is written in Symbol-Number Representation the flow of information through the translator, assembler and the loader can be represented as shown in Figure 5-1.

The direct translation process is the same as the code generation for the compilation of Fortran, Algol, or PL/1, only the source language is in Symbol-Number Representation. The translator also manipulates a list and its sublist. The list is the source language symbols list, and the sublist consists of the translated fragments in the object language which are symbolic instructions implemented by the computer manufacturers. In most cases the compiler will first translate the source language into an internal form which is easier to translate. Four internal forms are frequently used, they are Quadruples, Triples, Polish Notation and Trees. In the internal forms, operators and operands appear essentially in the order in which they are to be executed, which is helpful for subsequent analysis and code generation.

Code may be one of three forms,

1. Absolute instructions, which are placed in fixed locations and which the compiler can execute immediately.
2. An assembly language program which must then be assembled.
3. Machine language program placed on card images or on secondary storage (a binary deck, or object module) which must be linked

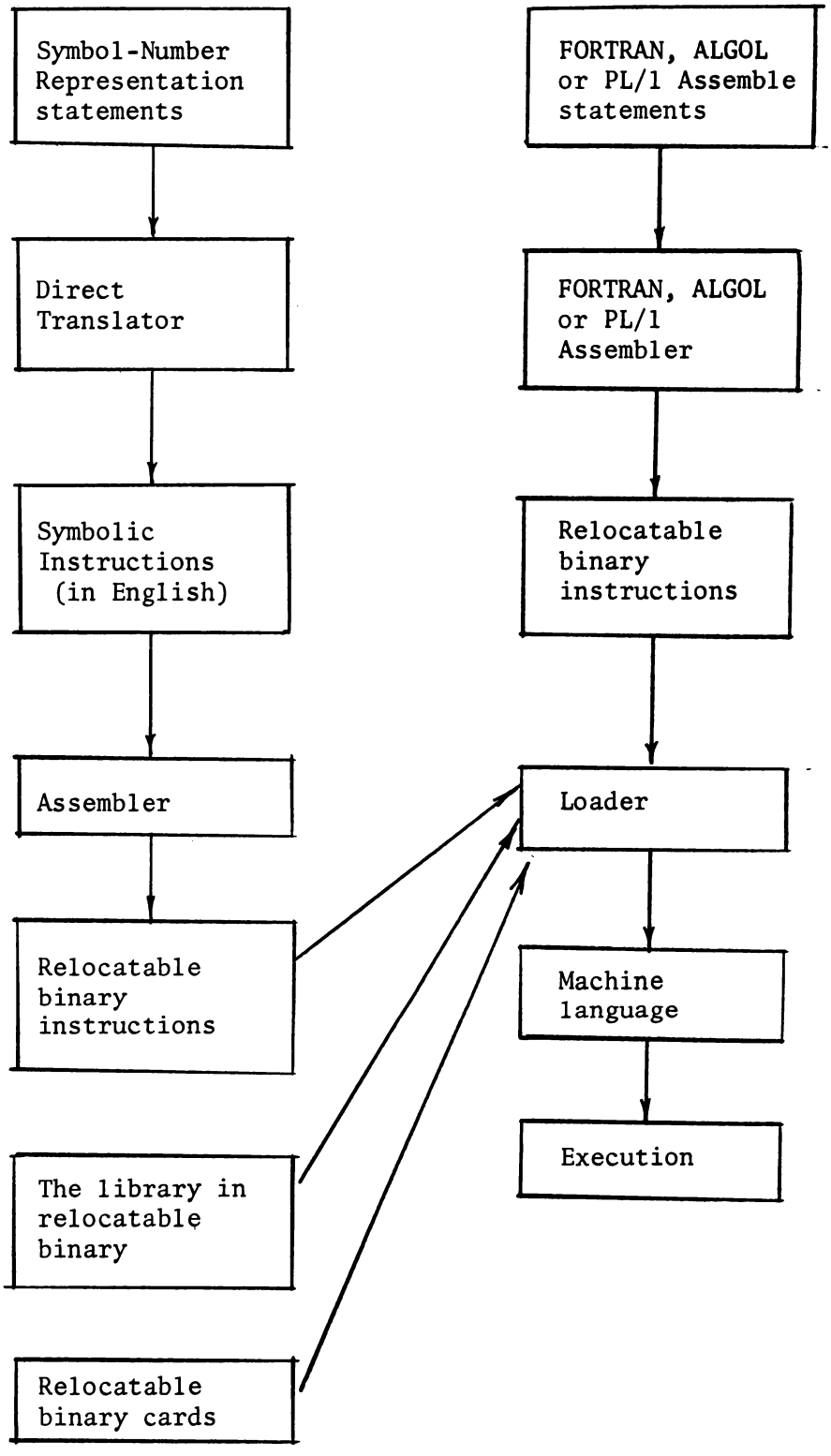


Figure 5-1. Overview of translation process.

with other subprograms and then loaded, in order to be executed.

The first method is the most efficient using time as the basis for comparison. The disadvantage is that one cannot easily precompile several subprograms separately and then link them together to execute. All the subprograms must be compiled at the same time. Producing assembly language is the easiest of the three methods, since one need not generate instructions as a sequence of bits, but it adds an extra step to the process of translating a program and this extra step often takes as long as the compilation itself.

Most commercial compilers produce an object module (binary deck) of the object program. This is a sequence of card images containing the machine language instructions.

In our case, we will only illustrate the generation of assembler code. The assembler code given in Table 5-1 is typical of those found on many machines.

In code generation, if we use the quadruple internal form, the description of temporary names is maintained throughout compile-time. With the triples, this is not necessary, since the triples require only three fields per entry. We need a description of each temporary name when using triples, but it only need to be maintained while code is being generated which can reference it. For our translation process, we will write a triple as (operand, operator, operand) instead of the regular form of (operator, operand, operand). For example, when we generate code for the expression A + B we write the internal form as

Table 5-1. List of typical assembler language instructions.

One address machine	Assembler codes	The machine provides with an accumulator (ACC) that always contain one operand of a binary operation. C(ACC) means the content of accumulator, and C(A) means the content of address A.
	ADD A	C(ACC) + C(A) stored as C(ACC)
	SUB A	C(ACC) - C(A) stored as C(ACC)
	MPY A	C(ACC) * C(A) stored as C(ACC)
	DIV A	C(ACC) / C(A) stored as C(ACC)
	LOAD A	C(A) stored as C(ACC)
	STORE A	C(ACC) to be stored as C(A)
	CLA B or CAD B	Clear and add, first clear the accumulator to zero and then adding the operand to it.
	RTP B	Raise C(ACC) to the power of C(B), leave the result in accumulator.
	LTER B	Load and test, C(B) stored as C(ACC) and test.
Zero address machine		The CPU of a machine does not require any addresses, it uses the stack principle (known as push-down store) TL is the top level of stack, SL is the second level of stack.
	LOAD A	Push C(A) stored as C(TL)
	STORE A	C(TL) stored as C(A)
	ADD A	C(SL) + C(TL) stored as C(SL)
	SUB A	C(SL) - C(TL) stored as C(SL)
	MPY A	C(SL) * C(TL) stored as C(SL)
	DIV A	C(SL) / C(TL) stored as C(SL)

(A + B). If we have more than one triple we generate a description of its result. The result of the ith triple is T(I), which will be referenced for the later triples. The code generators are similar to those used for quadruples or polish notation. The major differences are that more work must be performed to find the name corresponding to the triple number, to delete temporaries from the stack, and to generate the name of the result of each triple and put it on the stacks. The name of the temporary is always in one of the two top stack levels when code is being generated to reference it.

Table 5-2 is an illustration of using the triple internal form to translate the arithmetic expression A * B + C - C * D . Column 1 is the triple, column 2 shows the efforts of replacing operands which reference triples by the corresponding temporary names, and column 4 shows the stacks (I) and T(I) after processing the triple. (I) represents the number of the triple and T(I) the storing of the result of this triple.

The direct translation is not only a series of replacements of the source language symbols with their corresponding object symbols, as we already stated in Chapter IV. The translator still manipulates a systematic comparison of the source language symbols with the components of the syntax rules, then generates the proper assembler codes. It should be noted that the object codes generated at one level could not be copied at the next level, since some of the codes have to be changed. We will use the triple representation for the translation process, combined with the syntax rules for the regular arithmetic expression given in Table 5-3.

Table 5-2. Translation of A*B+C-C*D using triples.

triple	become	code	stack	ACC
(1) (A*B)	(A*B)	LOAD A MPY B	(1) T(1)	T(1)
(2) ((1)+C)	(T1+C)	ADD C STORE T(2)	(2) T(2)	T(2)
(3) (C*D)	(C*D)	LOAD C MPY D	(3) T(3)	T(3)
(4) ((2)-(3))	(T(2)-T(3))	STORE T(3) LOAD T(2) SUB T(3)	(4) T(4)	T(4)

Table 5-3. Syntax rules for regular arithmetic expressions.

1. $\langle v \rangle ::= \langle \text{variable name} \rangle$

2. $\langle eq \rangle ::= \dfrac{=}{(\text{empty})}$

3. $\langle p \rangle ::= \dfrac{**}{(\text{empty})}$

4. $\langle hop \rangle ::= \dfrac{*}{\text{MPY}} \mid \dfrac{/}{\text{DIV}}$

5. $\langle \text{lft par} \rangle ::= \dfrac{(}{(\text{empty})}$

6. $\langle lop \rangle ::= \dfrac{+}{\text{ADD}} \mid \dfrac{-}{\text{SUB}}$

7. $\langle \text{rit par} \rangle ::= \dfrac{)}{(\text{empty})}$

8. $\langle \text{lft pt} \rangle ::= \dfrac{\langle v \rangle \quad \langle eq \rangle}{\text{STORE} \quad v}$

9. $\langle t \rangle ::= \dfrac{\langle v \rangle}{\langle v \rangle} \quad \dfrac{\langle \text{lft par} \rangle \quad \langle hf \rangle \quad \langle \text{rit par} \rangle}{\langle t \rangle}$

10. $\langle f \rangle ::= \dfrac{\langle t \rangle}{\langle t \rangle} \quad \dfrac{\langle f \rangle \quad \langle p \rangle \quad \langle t \rangle}{\begin{array}{l}\text{LOAD} \quad \langle hf \rangle \\ \text{RIP} \quad \langle t \rangle \\ \text{STORE} \quad T(I)\end{array}}$

11. $\langle hf \rangle ::= \dfrac{\langle f \rangle}{\langle f \rangle} \quad \dfrac{\langle hf \rangle \quad \langle hop \rangle \quad \langle f \rangle}{\begin{array}{l}\text{LOAD} \quad \langle hf \rangle \\ \text{MPY} \quad \langle f \rangle \quad (\text{or DIV} \quad \langle f \rangle \quad) \\ \text{STORE} \quad T(I)\end{array}}$

12. $\langle lav \rangle ::= \dfrac{\langle hf \rangle}{\langle hf \rangle} \quad \dfrac{\langle lav \rangle \quad \langle lop \rangle \quad \langle hf \rangle}{\begin{array}{l}\text{LOAD} \quad \langle lav \rangle \\ \text{ADD} \quad \langle hf \rangle \quad (\text{or SUB} \quad \langle hf \rangle \quad) \\ \text{STORE} \quad T(I)\end{array}}$

13. $\left\langle \begin{array}{l}\text{assignment} \\ \text{statement}\end{array} \right\rangle ::= \dfrac{\langle \text{lft pt} \rangle \quad \langle lav \rangle}{\text{STORE} \quad \langle lav \rangle}$

As can be noted from Table 5-3, when the symbols of the source language fit rules 10, 11 and 12, we use the triple method to generate code. In order to do this the following four steps must be followed.

1. Scan the symbols from left to right, the first symbol S1 is an operand, we generate LOAD S1, or CLA S1.
2. The second symbol S2 is an operator, the third symbol is an operand. If S2 is "+" we generate ADD S3, if S2 is "*" we generate MPY S3 ... etc.
3. Next we generate STORE T(I), where I is the number of the triple and T(I) is the temporary. Add 1 to I for later code generation.
4. After we complete steps 1 to 3, if some higher level number code generation process must still be invoked, we have a choice of either copying the code or using T(I) as a variable.

The example in Figure 5-2 will illustrate this. From Figure 5-2, one can note that every vertical line indicates that a separate translation process occurs. The number above the upper end of the line is the order of the process, and the syntax rule numbers applied are on the right side of the line. The following steps are employed in this process.

1. By syntax rule 1 A is a <variable name> .
2. Rule 2 "=" is <eq> .
3. Rule 8 requires that a <variable name> followed by a "=" is <lft pt> .
4. Rule 1 IM is <v> .

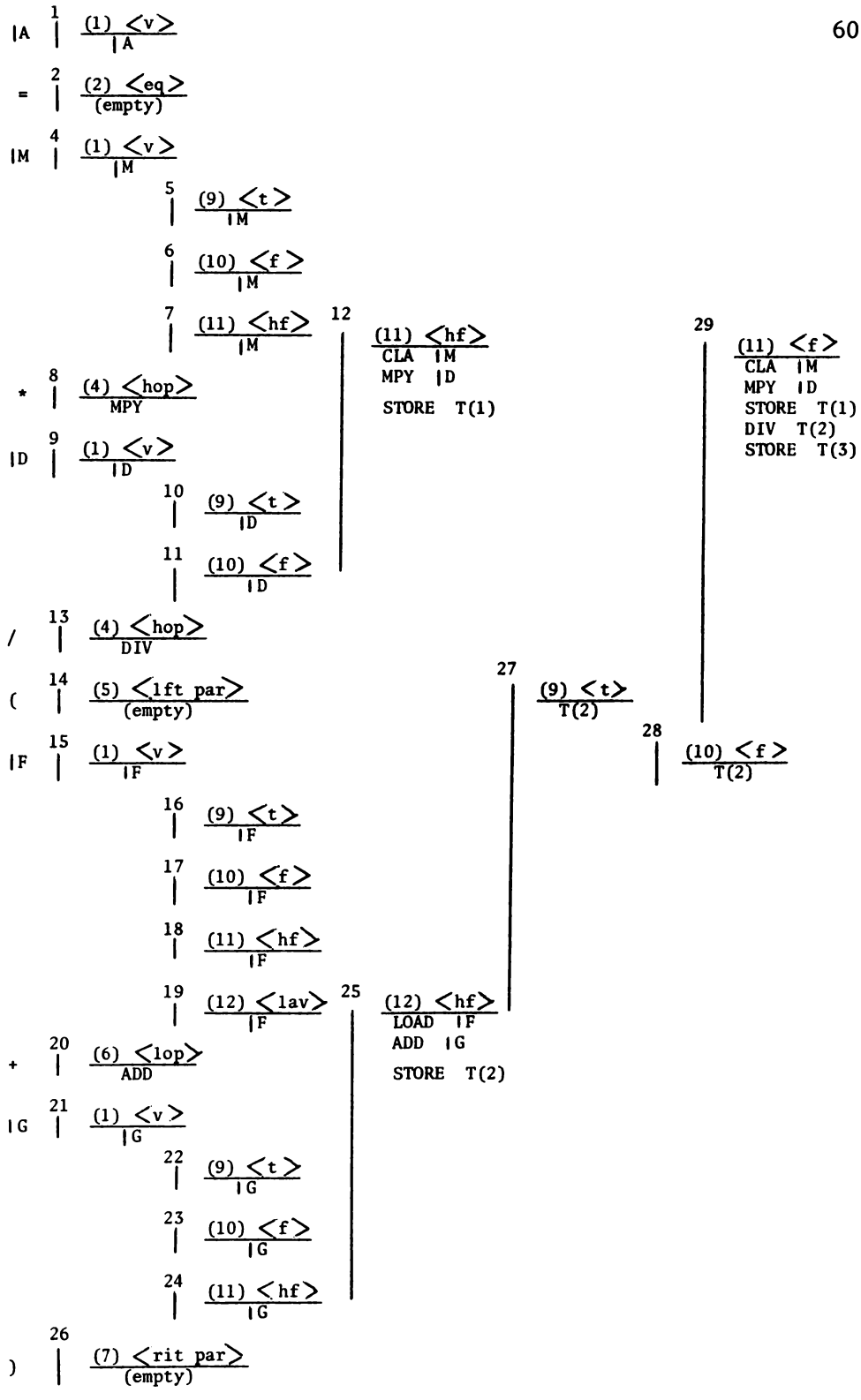


Figure 5-2. Compilation of |A = |M * |D / (|F + |G).

5. The operator just following |M is "/" which is a <hop> , so we have to process |M according to the syntax rules from top down until we find |M followed by <hop> . From rule 9, when we find <v> we change it to <t> .
6. From rule 10, <t> is transformed to <f> .
7. From rule 11, f to hf .
8. From rule 4, "/" to <hop> .
9. From rule 1, |D to <v> .
10. From rule 9, <v> to <t> .
11. From rule 10, <t> to <f> .
12. From rule 11, <hf> ::= <hf> <hop> <f>

Combining processes 7, 8, 11 we find that |M / |D is <hf> <hop> <f>, which is changed to <hf> . The codes generated are CLA |M, MPY |D, STORE T(I). Continuing in the same way, we obtain the complete translation.

We will illustrate this procedure with a second example. We use a conditional statement RW04446 (E) n_1, n_2, n_3 where E is an arithmetic expression, which has to be evaluated with our triple method and stored in temporary T(I), and N_1, n_2, and n_3 are integers. Several registers and an accumulator are enough to execute this conditional statement. Figure 5-3 shows the steps involved in translating the expression |RW04446 (B-C) 2, 3, 2, and the syntactic rules for expressing a conditional statement are given in Table 5-4.

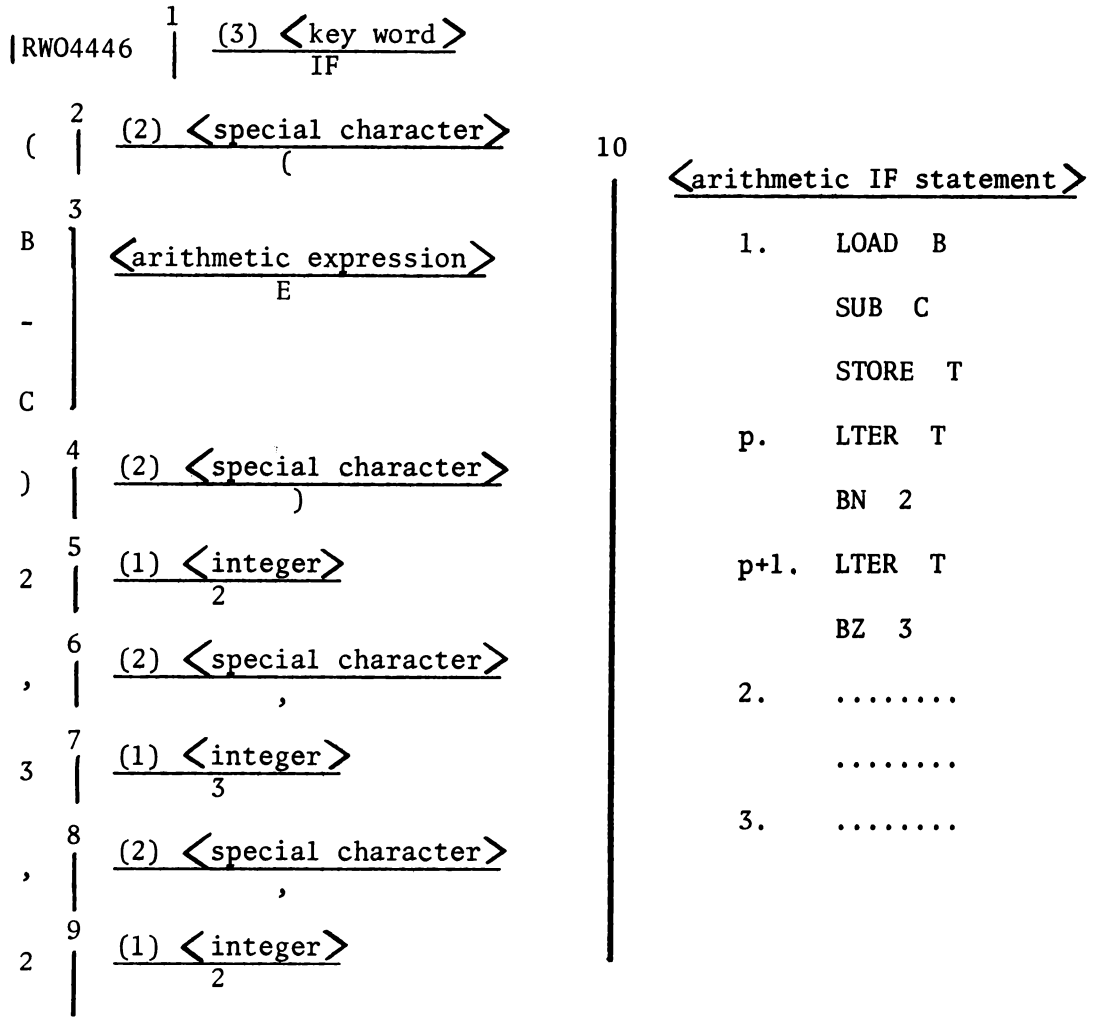


Figure 5-3. Translation of conditional statement |RWO4446 (B-C) 2,3,2.

Table 5-4. Rules for evaluating a conditional statement.

1. <integer> ::= <digit>^{n}
2. <special character> ::= (|) | ,
3. <key word> ::= <SN-word>
4. <arithmetic IF statement> ::= <key word> <special character> <arithmetic expression> <special character> <integer> <special character> <integer> <special character> <integer>

Generating codes for a conditional statement are much different than for an arithmetic expression. We first need to transform the statement into triples as follows.

```
1.  TRIPLE for  T =    E
    .....................
    p.      (TEST1  2,  T)
    p+1.    (TEST2  3,  T)
2.  .....................
    .....................
3.  .....................
```

where "TRIPLE for T = E " is a routine which will transform the arithmetic expression into triples and generate proper codes. The codes are LOAD B , SUB C , STORE T. After this triple, we have (TEST1 2, T) TEST1 is a routine which will test the content of the temporary T. If the content of T is negative the execution branches to the location named 2. If the content of T is not negative, the execution will proceed to the next step which is the triple (TEST2 3, T). TEST2 is a routine which will test the content of T. If the content of T is zero, the execution branches to the location named 3. If the content of T is not zero, the execution will go to the next step which is the location named 2. The triple (TEST1 2, T) generate the codes LTER T, BN 2. The triple (TEST2 3, T) generates the codes LTER T, BZ 3.

CHAPTER VI

CONCLUSION

The author proposed the method of Symbol-Number Representation in order to accomplish the following goals:

1. To not store any words in memory for input-output references. (Since for regular documents we have to prepare about 3000 words which would occupy at least 50,000 bytes).
2. To input Chinese words without using any references.
3. To transmit Chinese characters without any difficulties during execution time.
4. To use the Symbol-Number Representation signal to control the printer in order to output the words with their correct shape.

In January, 1974, a group of researchers held a meeting at the EE Department at Taiwan University in Taipei. This meeting was sponsored by the Committee of National Sciences with the purpose of discussing how to implement the plan of building the Chinese computer. These researchers came from four institutions which were undertaking similar research projects. All of these research projects based their research on storing the words in memory. Taiwan University has stored 8,000 words in memory. Tsing Hua University uses a 32 32 dot matrix to store the whole words. Chiao Tung University uses a 24 24 dot matrix to store 496 elementary strokes in memory.

From our simulated examples we can see that we can carry out goals 1, 2, and 3 without much difficulty, since we did not store any words in memory. (This means we did not use the dot matrix method to store words for reference purposes.)

As to the fourth goal, the author believes the Symbol-Number Representation can be used to control the printer to print the original words, as in Extended Binary Coded Decimal Interchange Code System where 11000001 orders the printer to print out "A", and 11000010 orders the printer to print out "B". In our example, we would like |DWO 0468 (10011110 11011111 11110000 11110100 11110110 11111000) to initiate the printer to print out the word 讀 (to read), and SYE 3032 (10101011 11011110 11001010 11110011 11110000 11110110 11110010) to initiate the printer to print out the word 寫 (to write). The author has attempted to show that this might be possible. At present we can not verify our simulated examples since such a printer has not been built. However, some results are given in Appendices A and B which summarize the author's attempts to build a translator using the Symbol-Number Representation. These initial results indicate further research should be conducted in this area.

BIBLIOGRAPHY

BIBLIOGRAPHY

1. Chieh, C. C. "The Analysis of the Elementary Strokes of Chinese Words," Research Reports of Chiao Tung University, Volume 6 (April, 1973).

2. Chieh, C. C. "How to Store the Elementary Strokes and to Synthesize the Words," Research Reports of Chiao Tung University, Volume 6 (April, 1973).

3. Cheathem, T. E. "The TGS-II Translator System," IFIP (1965).

4. Cheathem, T. E., and Sattley, K. "Syntax Directed Compiling," SJCC (1964).

5. Dijkstra, E. W. "Making a Translator for Algol 60," APIC Bull. 7 (May 1961).

6. Gear, C. William. Computer Organization and Programming, McGraw-Hill Book Company (1969).

7. Gries, David. Compiler Construction for Digit Computers, Cornell University, John Wiley & Sons, Inc. New York (1971).

8. Reifler, Erwin. Machine Translation, New York, John Wiley & Sons, Inc. (1967).

9. Hassitt, Anthony. Computer Programming and Computer System, Academic Press, New York (1963).

10. Irons, E. T. "A Syntax Directed Computer for Algol 60," Communication of the ACM (Jan. 1961).

11. Ingerman, P. Z. "A Syntax Oriented Translator," Academic Press, New York (1966).

12. Lee, John A. N. The Anatomy of a Compiler, Reinhold, New York (1967).

13. Ledley, Robert S., and Wilson, James B. "Automatic Programming Language Translation through Syntactical Analysis," Communications of the ACM (1967).

14. Lo, Chao Chien. "A Recognition Routine for Chinese Sentences," Master's Thesis, University of Tennessee (1971).

15. McEwan, A. T. "An Atlas Autocode to Algol 60 Translator," Computer Journal, Volume 9 (May 1966-Feb. 1967).

16. Morris, D., and Rohl, J. S. "Trees and Routine," Comp. Journal 5, (April, 1962).

17. Randell, B., and Russell, L. J. "Single-Scan Techniques for the Translation of Arithmetic Expression in the Algol 60," Journal of the Association for Computing Machinery, Vol. 11, No. 2 (April 1964).

18. Reifler, Erwin. Machine Translation, New York, John Wiley & Sons, Inc. (1967).

APPENDIXES

APPENDIX A

SCANNER

```
      DIMENSION OUTVAR(20)
      REAL*8 OUTVAR
      REAL A,D,E,G,L,N,O,Q,R,T
      REAL BLA,LPA,RPA,PLU,MIN,AST,SLA,COM,PER,EQU,VER
      DATA BLA/' '/,LPA/'('/,RPA/')'/,PLU/'+'/,MIN/'-'/,AST/'*'/,
     1SLA/'/'/,COM/','/,PER'.'/,EQU/'='/,VER/' '/
      DATA A/'A'/,D/'D'/,E/'E'/,G/'G'/,L/'L'/,N/'N'/,O/'O'/,Q/'Q'/,
     1R/'R'/,T/'T'/
      REAL SOU(80),VAR(8)
      INTEGER SHIER(80),V,W
      DEFINE FILE 8(80,8,E,V)
   11 DO 297 I=1,20
  297 WRITE(8'I,298)
  298 FORMAT(8H        )
   81 DO 82 J=1,80
      SOU(J)=BLA
   82 SHIER(J)=BLA
   83 READ(5,90,END=500) SOU
   90 FORMAT(80A1)
      PRINT 93
   93 FORMAT(//1HO,'INITIAL INPUT STATEMENT:')
   91 WRITE(6,92) SOU
   92 FORMAT(1HO,80A1)
  400 J=1
      I=1
  401 IF  (SOU(J).EQ.RPA.OR.SOU(J).EQ.EQU.OR.SOU(J).EQ.MIN.OR.SOU(J).EQ.
     1PLU.OR.SOU(J).EQ.AST.OR.SOU(J).EQ.COM.OR.SOU(J)
     1.EQ.PER.OR.SOU(J).EQ.SLA.OR.SOU(J).EQ.LPA) GO TO 190
      IF(SOU(J).EQ.BLA) GO TO 192
      IF(SOU(J).EQ.VER) GO TO 667
      SHIER(J)=2
      GO TO 185
  667 SHIER(J)=3
      GO TO 185
  192 SHIER(J)=0
      GO TO 185
  190 SHIER(J)=1
  185 J=J+1
      IF(J.LE.80) GO TO 401
      J=1
      GO TO 5
    4 J=J+1
    5 IF(SHIER(J).EQ.2) GO TO 3
      IF(SOU(J) .EQ. VER) GO TO 113
      IF(SOU(J).EQ.MIN) GO TO 101
      IF(SOU(J).EQ.AST) GO TO 249
      IF(SOU(J).EQ.BLA) GO TO 250
```

```
 10 IF(SOU(J).EQ.PER) GO TO 100
    GO TO 600
100 J=J+1
    IF(SHIER(J).EQ.2) GO TO 25
    GO TO 500
 25 IF(SOU(J).EQ.E.OR.SOU(J).EQ.G.OR.SOU(J).EQ.L.OR.SOU(J).EQ.N
   1.OR.SOU(J).EQ.A.OR.SOU(J).EQ.O ) GO TO 26
    J=J-1
    GO TO 3
447 DO 597 K=3,8
597 VAR(K)=BLA
    VAR(1)=AST
    VAR(2)=AST
    GO TO 40
249 J=J+1
    IF(SOU(J).EQ.AST) GO TO 447
    J=J-1
    GO TO 600
113 DO 49 K=2,8
 49 VAR(K)=BLA
    VAR(1)=SOU(J)
    K=1
    J=J+1
111 1F(SHIER(J).EQ.@.OR.SOU(J).EQ.VER) GO TO 114
    IF(SOU(J).EQ.BLA) GO TO 40
    IF(SHIER(J).EQ.L) GO TO 333
    IF(K.EQ.8) GO TO 333
    GO TO 500
333 J=J-1
    GO TO 40
114 K=K+1
    VAR(K)-SOU(J)
    J=J+1
    GO TO 111
 26 J=J+1
    IF(SOU(J).EQ.T.OR.SOU(J).EQ.E.OR.SOU(J).EQ.Q.OR.SOU(J).EQ.N
   1.OR.SOU(J).EQ.R.OR.SOU(J).EQ.)) GO TO 27
    GO TO 500
 27 J=J+1
    IF(SOU(J).EQ.PER) GO TO 39
    IF(SOU(J).EQ.T.OR.SOU(J).EQ.D) GO TO 28
    GO TO 500
 28 J=J+1
    IF(SOU(J).EQ.PER) GO TO 38
    GO TO 500
101 J=J-1
    IF(SHIER(J).EQ.2) GO TO 103
    IF(SOU(J).EQ.RPA) GO TO 103
    J=J+1
    DO 599 K-4,8
```

```
599 VAR(K)-BLA
    VAR(1)=N
    VAR(2)=E
    VAR(3)=G
    GO TO 40
  3 DO 6 K=2,8
  6 VAR(K)=BLA
    VAR(1)=SOU(J)
    J=J+1
    K=1
  7 IF(SHIER(J).EQ.@) GO TO 9
    IF(SOU(J).EQ.PER) GO TO 18
    J=J-1
    GO TO 40
 38 J=J-4
    GO TO 800
 39 J=J-3
    GO TO 700
103 J=J+1
    GO TO 600
 18 J=J+1
    IF(SOU(J).EQ.E.OR.SOU(J).EQ.G.OR.SOU(J).EQ.L.OR.SOU(J).EQ.N
   1.OR.SOU(J).EQ.A.OR.SOU(J).EQ.O ) GO TO 19
    J=J-1
  9 K=K+1
    VAR(K)=SOU(J)
    J=J+1
    GO TO 7
700 DO 16 K=2,8
 16 VAR(K)=BLA
    VAR(1)=SOU(J)
    K=1
    DO 17 M=1,3
    J=J+1
    K=K+1
 17 VAR(K)=SOU(J)
    GO TO 40
600 DO 601 K=2,8
601 VAR(K)=BLA
    VAR(1)=SOU(J)
    GO TO 40
800 DO 801 K=2,8
801 VAR(K)=BLA
    VAR(1)=SOU(J)
    K=1
    DO 802 M=1,4
    J=J+1
    K=K+1
802 VAR(K)=SOU(J)
    GO TO 40
```

```
 19 J=J-2
 40 WRITE(8'I,102) VAR
102 FORMAT(8A1)
    I=I+1
    GO TO 4
250 J=J+1
    IF(SOU(J).EQ.BLA) GO TO 501
    GO TO 5
501 PRINT 252
252 FORMAT(//1HO,'SYMBOL TABLE AFTER PASSING SCANNER:')
    DO 310 M=1,I
307 READ (8'M,308) OUTVAR(M)
308 FORMAT(A8)
    WRITE(6,309) OUTVAR(M)
309 FORMAT(1HO,4X,A8)
310 CONTINUE
    GO TO 11
500 STOP
    END
```

INITIAL INPUT STATEMENT:

1RW04446(B-A) 2,3,2

SYMBOL TABLE AFTER PASSING SCANNER:

RW04446

(

B

-

A

)

2

,

3

,

2

APPENDIX B

PRIMARY STEP OF RECOGNITION

OF STATEMENT

```
    DIMENSION STK(20),NAM(8,10)
    REAL*8  IDE,VAR,IND,LET,DIG,FAC,CON,TER,SNW,SYNA,STK,EMPTY
   1,ID,CO
    INTEGER PLU,MIN,AST,DIV,DEX,COM,PER,SEM,APO,EQU,BLA,DOR,VER,LPA,
   1RPA,NAM,NUM,ZER,NIN,A,Z,V,E
    DATA SYNA/' SYNNAM '/,IDE/' IDENTI '/,VAR/' VARIAB '/,
   1LET/' LETTER '/,DIG/' DIGIT '/,CON/' CONSTA '/,SNW/' SNWORD '/
    DATA ZER/'0'/,NIN/'9'/,A/'A'/,Z/'Z'/,EMPTY/' '/,DIV/'/'/,VER/' '/,
   1LPA/'('/,RPA/')'/,NUM/'#'/,DEX/'**'/,AST/'*'/,MIN/'='/,PLU/'+'/,
   1COM/','/,PER/'.'/,BLA/' '/,SEM/':'/,APO/''''/,EQU/'='/,DOR/'$'/
    DATA ID/'     I  '/,CO/'     C  '/
107 DO 103  I=1,20
103 STK(I)=EMPTY
    DEFINE FILE 8(80,8,E,V)
 11 DO 297 I=1,20
297 WRITE (8'I,298)
298 FORMAT(8H        )
    DO 20 J=1,10
    DO 21 I=1,8
 21 NAM(I,J)=BLA
 20 CONTINUE
    READ(5,99,END=500)((NAM(I,J),I=1,8),J=1,10)
 99 FORMAT(80A1)
    PRINT 400
400 FORMAT(1HO,3X,'INPUT STRING:')
    WRITE(6,108)((NAM(I,J),I=1.8),J=1,10)
108 FORMAT(1HO,3X,10(8A1))
    DO 102 J=1,10
    I=1
    IF(NAM(I,J).EQ.BLA) GO TO 102
 40 IF(NAM(I,J).EQ.VER) GO TO 39
    GO TO 86
 39 I=I+1
    IF(NAM(I,J).GE.A.AND.NAM(I,J).LE.Z) GO TO 39
    K=0
 35 IF(NAM(I,J).GE.ZER.AND.NAM(I,J).LE.NIN) GO TO 36
    IF(NAM(I,J).EQ.VER) GO TO 39
    IF(NAM(I,J).NE.BLA) GO TO 420
 34 IF(K.EQ.4) GO TO 31
    GO TO 32
420 PRINT 421
421 FORMAT(4X,'ERROR')
    GO TO 102
 32 WRITE(8'J,202) SYNA
202 FORMAT(A8)
    GO TO 102
 36 K=K+1
```

```
   I=I+1
   IF(I.GT.8) GO TO 32
   GO TO 35
31 WRITE(8'J,202) SNW
   GO TO 102
86 I=1
   IF(NAM(2,J).EQ.BLA) GO TO 78
   IF(NAM(I,J).GE.A.AND.NAM(I,J).LE.Z) GO TO 87
   GO TO 83
87 I=I+1
   IF(NAM(I,J).GE.A.AND.NAM(I,J).LE.Z .OR.NAM(I,J).GE.ZER.AND.
  1NAM(I,J).LE.NIN) GO TO 87
   IF(NAM(I,J).EQ.BLA.OR.I.GT.6) GO TO 84
   GO TO 420
84 WRITE (8'J,202) IDE
   GO TO 102
83 IF(NAM(1,J).GE.ZER.AND.NAM(1,J).LE.NIN.OR.NAM(1,J).EQ.PER.OR.
  1NAM(1,J).EQ.MIN) GO TO 82
   GO TO 72
82 I=I+1
80 IF(NAM(I,J).GE.ZER.AND.NAM(I,J).LE.NIN.OR.NAM(I,J)).EQ.PER) GO TO 82
   IF(NAM(I,J).EQ.BLA.OR.I.EQ.8) GO TO 92
   GO TO 420
78 IF(NAM(1,J).GE.ZER.AND.NAM(1,J).LE.NIN) GO TO 52
   IF(NAM(1,J).GE.A.AND.NAM(1,J).LE.Z) GO TO 53
   GO TO 71
72 IF(NAM(2,J).EQ.AST) GO TO 67
   GO TO 420
67 IF(NAM(1,J).EQ.AST) GO TO 54
   GO TO 500
71 IF(NAM(1,J).EQ.PLU) GO TO 65
   IF(NAM(1,J).EQ.MIN) GO TO 64
   IF(NAM(1,J).EQ.AST) GO TO 63
   IF(NAM(1,J).EQ.DIV) GO TO 62
   IF(NAM(1,J).EQ.PER) GO TO 61
   IF(NAM(1,J).EQ.COM) GO TO 60
   IF(NAM(1,J).EQ.LPA) GO TO 59
   IF(NAM(1,J).EQ.RPA) GO TO 58
   IF(NAM(1,J).EQ.EQU) GO TO 57
   IF(NAM(1,J).EQ.SEM) GO TO 56
   IF(NAM(1,J).EQ.APO) GO TO 55
   GO TO 420
92 WRITE(8'J,202) CON
   GO TO 102
65 WRITE(8'J,202) PLU
   GO TO 102
64 WRITE(8'J,202) MIN
   GO TO 102
```

```
 63 WRITE(8'J,202) AST
    GO TO 102
 62 WRITE(8'J,202) DIV
    GO TO 102
 61 WRITE(8'J,202) PER
    GO TO 102
 60 WRITE(8'J,202) COM
    GO TO 102
 59 WRITE(8'J,202) LPA
    GO TO 102
 58 WRITE(8'J,202) RPA
    GO TO 102
 57 WRITE(8'J,202) EQU
    GO TO 102
 56 WRITE(8'J,202) SEM
    GO TO 102
 55 WRITE(8'J,202) APO
    GO TO 102
 54 WRITE(8'J,202) DEX
    GO TO 102
 53 WRITE(8'J,202) LET
    GO TO 102
 52 WRITE(8'J,202) DIG
102 CONTINUE
    DO 310 M=1,10
310 READ(8'M,308) STK(M)
308 FORMAT(A8)
    PRINT 401
401 FORMAT(1HO,3X,'RECOGNIZED AS:')
    WRITE(6,309)(STK(M),M=1,10)
309 FORMAT(1HO,3X,10A8)
    DO 402 I=1,20
    IF(STK(I).EQ.WYNA.OR.STK(I).EQ.IDE.OR.STK(I).EQ.LET) GO TO 403
    IF(STK(I).EQ.DIG.OP..STK(I).EQ.CON) GO TO 404
    GO TO 402
403 STK(I)=EMPTY
    STK(I)=ID
    GO TO 402
404 STK(I)=EMPTY
    STK(I)=CO
402 CONTINUE
    PRINT 405
405 FORMAT(1HO,3X,'SIMPLIFIED AS:')
    WRITE (6,406) STK
406 FORMAT(1HO,3X,10A8////)
    GO TO 107
500 STOP
    END
```

INPUT STRING:

|AD45 |AH1234 |AC009 |AA990

RECOGNIZED AS:

<SYNNAM><SNWORD><SYNNAM><SYNNAM>

SIMPLIFIED AS:

I <SNWORD> I I

INPUT STRING:

|AD1357 (5 , 26) ADE

RECOGNIZED AS:

<SNWORD> (<DIGIT> , <CONSTA>) <IDENTI>

SIMPLIFIED AS:

<SNWORD> (C , C) I

INPUT STRING:

ADFRE = |AR35 + XY * 75

RECOGNIZED AS:

<IDENTI> = <SYNNAM> + <IDENTI> * <CONSTA>

SIMPLIFIED AS:

I = I + I * C

INPUT STRING:

DER ** |EI S34 - 3.9

RECOGNIZED AS:

<IDENTI> ** <SYNNAM> - <CONSTA>

SIMPLIFIED AS:

I ** I - C

VITA

Bill Wei-sung Chang was born on June 9, 1929 in Tsingtao, China. He received his B.S. degree in Mechanical Engineering in 1954 from the Chinese Naval College of Technology, Tsoying, Taiwan.

Finishing undergraduate school, he accepted employment with Chinese Naval 3d shipyard at Keelung as an engineer.

In March, 1970 he came to the University of Tennessee as a graduate student to work on his master's degree in the department of computer science. In May, 1974 he received a Master of Science degree with a major in computer science from the University of Tennessee.